SIDAC-I is a digital analog simulator which permits analog as well as hybrid simulations to be programmed on a digital computer. FORTRAN, a scientific computer language, may be included as an integral part of the SIDAC-I simulation language. The simulator consists of a controller program and a set of predefined functional subroutines representing the standard analog computer elements and logical components. Being aware of having the simulator written for a relatively small computer, all the predefined functional subroutines were designed to be independent of the main program and other functional subprograms so that only the functions which are required to solve a certain problem have to be loaded into the computer. This technique of simulation provides not only time saving opportunity, but more space in the core memory for larger simulation problems as well.

To implement accurate integration with a high degree of
stability SIDAC-I uses the second-order Runge-Kutta integration method. It includes a capability and high degree of flexibility for modifying the parameters and initial conditions. In addition to this, SIDAC-I is capable of accepting some additional computation time to carry on the computation from a point starting with the previous termination time. Although SIDAC-I does not provide the outstanding man-machine rapport all the desirable features that it has will take some of the burden from the man-machine interface.
SIDAC-I A SIMULATED DIGITAL-ANALOG COMPUTER

by

MANOO ORDEEDOLCHEST

A THESIS

submitted to

OREGON STATE UNIVERSITY

in partial fulfillment of
the requirements for the
degree of

MASTER OF SCIENCE

June 1966
APPROVED:

Professor of Electrical Engineering
In Charge of Major

Head of Department of Electrical Engineering

Dean of Graduate School

Date thesis is presented May 19, 1966

Typed by Marion F. Palmateer
ACKNOWLEDGEMENTS

My most sincere appreciation is extended to Professor Louis N. Stone for his encouragement and advice throughout the course of this study and for his help in the preparation of this thesis.

Thanks go to the Staff of the Department of Electrical Engineering particularly Dr. John L. Saugen, and to the graduate students of this department for their advice and willingness to discuss topics of interest.

Thanks are also due to Mr. Robert D. Brennan of the IBM Research Laboratory at San Jose for sending the source deck of his PACTOLUS simulator program and some valuable information on digital simulation.

I wish to express my gratitude to Mr. David G. Niess of the O. S. U. Statistics Computing Laboratory for his instruction as well as the technical assistance given while this research work was being conducted on the IBM 1620 computer. A special debt of gratitude is due to David C. Squire who critically read the manuscript.

The Institute of International Education and the Oregon State Foreign Scholarship Commission are also gratefully acknowledged for their financial support.
TABLE OF CONTENTS

INTRODUCTION

THE GENERAL DESCRIPTION OF THE SIDAC-I SIMULATOR

The Technique of Simulation

THE DESCRIPTION OF THE SIDAC-I ELEMENTS

Automatic Stop 10
AND Gate 10
Bang-Bang 11
Clock 12
Comparator 13
Counter 13
Dead Space 14
Delay 15
Flipflop 16
Function Generator 17
Integrator 20
Limiter 33
NAND Gate 34
Negative Clipper 35
NOR Gate 35
NOT Gate 36
Offset 36
OR Gate 36
PARAMETER 37
PLOTER 38
Positive Clipper 42
Pulse Generator 42
Punch-output 43
RANDOM Noise Generator 43
Relay 45
Sawtooth Generator 46
Square-Wave Generator 47
Track-Transfer 48
Type-Output 49
Zero-Order Hold 49
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIDAC-I COMPUTER CODING</td>
<td>53</td>
</tr>
<tr>
<td>Program Structure and Its Composition</td>
<td>53</td>
</tr>
<tr>
<td>Procedure of Setting Additional Runs and Reinitialization</td>
<td>64</td>
</tr>
<tr>
<td>Procedure for Interrupting the Run</td>
<td>66</td>
</tr>
<tr>
<td>Deck Arrangement and Compilation Procedure</td>
<td>67</td>
</tr>
<tr>
<td>Sense Switches</td>
<td>70</td>
</tr>
<tr>
<td>Error Messages</td>
<td>71</td>
</tr>
<tr>
<td>EXAMPLES</td>
<td>76</td>
</tr>
<tr>
<td>Example 1: SIDAC-I Solution of the Van der Pol Equation</td>
<td>76</td>
</tr>
<tr>
<td>Example 2: Hybrid Solution of a Optimization Control Problem</td>
<td>79</td>
</tr>
<tr>
<td>Example 3: Hybrid Solution of a Optimization Control Problem</td>
<td>82</td>
</tr>
<tr>
<td>CONCLUSIONS</td>
<td>92</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>94</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>96</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>96</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>97</td>
</tr>
<tr>
<td>APPENDIX C</td>
<td>112</td>
</tr>
<tr>
<td>APPENDIX D</td>
<td>114</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>Transfer characteristics of a Bang-Bang.</td>
</tr>
<tr>
<td>2</td>
<td>Transfer characteristics of a comparater.</td>
</tr>
<tr>
<td>3</td>
<td>An example illustrating the input and output signals of a zero crossing counter.</td>
</tr>
<tr>
<td>4</td>
<td>The transfer characteristics of the Dead-Space.</td>
</tr>
<tr>
<td>5</td>
<td>The control pulse waveforms in one integration period.</td>
</tr>
<tr>
<td>6</td>
<td>The geometrical representation of the second-order Runge-Kutta numerical integration method.</td>
</tr>
<tr>
<td>7</td>
<td>An electronic mode controlled integrator.</td>
</tr>
<tr>
<td>8</td>
<td>Illustration of a memory-pair operation.</td>
</tr>
<tr>
<td>9</td>
<td>The transfer characteristic of SIDAC-I limiter.</td>
</tr>
<tr>
<td>10</td>
<td>The SIDAC-I plot frame with the fixed grid size.</td>
</tr>
<tr>
<td>11</td>
<td>Offset elements and gain constants are used to assist the scaling of x - y coordinates.</td>
</tr>
<tr>
<td>12</td>
<td>Pulses generated by SIDAC-I pulse generator.</td>
</tr>
<tr>
<td>13</td>
<td>The pictorial representation of SIDAC-I Relay.</td>
</tr>
<tr>
<td>14</td>
<td>The pictorial sawtooth wave of SIDAC-I sawtooth generator.</td>
</tr>
<tr>
<td>15</td>
<td>The typical square wave generated by SIDAC-I Square-Wave Generator.</td>
</tr>
<tr>
<td>16</td>
<td>Two cascade SIDAC-I Track-Transfers forming a wave shaping element.</td>
</tr>
<tr>
<td>17</td>
<td>SIDAC-I Zero-Order Hold operation in the time domain.</td>
</tr>
<tr>
<td>18</td>
<td>A complete deck of the SIDAC-I program.</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>19</td>
<td>Block diagram for the Van der Pol equation.</td>
</tr>
<tr>
<td>20</td>
<td>The typewriter output for the Van der Pol equation problem.</td>
</tr>
<tr>
<td>21</td>
<td>Graphical output for the Van der Pol equation problem.</td>
</tr>
<tr>
<td>22</td>
<td>Simulation diagram for optimization feedback control problem.</td>
</tr>
<tr>
<td>23</td>
<td>The typewriter output for Example 2.</td>
</tr>
<tr>
<td>24</td>
<td>The graphical output for Example 2.</td>
</tr>
<tr>
<td>25</td>
<td>Block diagram for Example 3.</td>
</tr>
<tr>
<td>26</td>
<td>The typewriter output for Example 3.</td>
</tr>
<tr>
<td>27</td>
<td>The graphical output for Example 3.</td>
</tr>
</tbody>
</table>

**Appendix**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-1</td>
<td>Block diagram of the SIDAC-I Main Control.</td>
<td>137</td>
</tr>
<tr>
<td>D-2</td>
<td>Block diagram of the SIDAC-I Automatic Stop.</td>
<td>140</td>
</tr>
<tr>
<td>D-3</td>
<td>Block diagram of the SIDAC-I AND Gate.</td>
<td>140</td>
</tr>
<tr>
<td>D-4</td>
<td>Block diagram of the SIDAC-I Bang-Bang.</td>
<td>141</td>
</tr>
<tr>
<td>D-5</td>
<td>(a) Block diagram of SIDAC-I Clock.</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>(b) Block diagram of SIDAC-I Comparator.</td>
<td>141</td>
</tr>
<tr>
<td>D-6</td>
<td>Block diagram of SIDAC-I Counter.</td>
<td>142</td>
</tr>
<tr>
<td>D-7</td>
<td>Block diagram of SIDAC-I Dead Space.</td>
<td>143</td>
</tr>
<tr>
<td>D-8</td>
<td>Block diagram of SIDAC-I Delay.</td>
<td>144</td>
</tr>
<tr>
<td>D-9</td>
<td>Block diagram of SIDAC-I Flipflop.</td>
<td>145</td>
</tr>
<tr>
<td>D-10</td>
<td>Block diagram of the SIDAC-I Function Generator.</td>
<td>146</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>D-11</td>
<td>Block diagram of the SIDAC-I Mode Controlled Integrator.</td>
<td>148</td>
</tr>
<tr>
<td>D-12</td>
<td>Block diagram of the SIDAC-I Limiter.</td>
<td>150</td>
</tr>
<tr>
<td>D-13</td>
<td>Block diagram of the SIDAC-I NAND Gate.</td>
<td>150</td>
</tr>
<tr>
<td>D-14</td>
<td>(a) Block diagram of the SIDAC-I Negative-Clipper.</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>(b) Block diagram of the SIDAC-I NOT Gate</td>
<td>151</td>
</tr>
<tr>
<td>D-15</td>
<td>Block diagram of the SIDAC-I NOR Gate.</td>
<td>151</td>
</tr>
<tr>
<td>D-16</td>
<td>Block diagram of the SIDAC-I OR Gate.</td>
<td>152</td>
</tr>
<tr>
<td>D-17</td>
<td>(a) Block diagram of the SIDAC-I Offset.</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td>(b) Block diagram of the SIDAC-I Positive-Clipper.</td>
<td>152</td>
</tr>
<tr>
<td>D-18</td>
<td>Block diagram of the SIDAC-I PARAMETER.</td>
<td>153</td>
</tr>
<tr>
<td>D-19</td>
<td>Block diagram of the SIDAC-I PLOTER.</td>
<td>154</td>
</tr>
<tr>
<td>D-20</td>
<td>Block diagram of the SIDAC-I Punch-Output.</td>
<td>155</td>
</tr>
<tr>
<td>D-21</td>
<td>Block diagram of the SIDAC-I Pulse Generator.</td>
<td>156</td>
</tr>
<tr>
<td>D-22</td>
<td>Block diagram of the SIDAC-I Random Noise Generator.</td>
<td>157</td>
</tr>
<tr>
<td>D-23</td>
<td>Block diagram of the SIDAC-I Relay.</td>
<td>157</td>
</tr>
<tr>
<td>D-24</td>
<td>Block diagram of the SIDAC-I Simple Integrator.</td>
<td>158</td>
</tr>
<tr>
<td>D-25</td>
<td>Block diagram of the SIDAC-I Square-Wave Generator.</td>
<td>159</td>
</tr>
<tr>
<td>D-26</td>
<td>Block diagram of the SIDAC-I Sawtooth Generator.</td>
<td>160</td>
</tr>
<tr>
<td>D-27</td>
<td>Block diagram of the SIDAC-I Type-Output.</td>
<td>161</td>
</tr>
<tr>
<td>D-28</td>
<td>Block diagram of the SIDAC-I Track-Transfer.</td>
<td>162</td>
</tr>
<tr>
<td>D-29</td>
<td>Block diagram of the SIDAC-I Zero-Order Hold.</td>
<td>163</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>The Truth Table for AND Gate.</td>
<td>11</td>
</tr>
<tr>
<td>II</td>
<td>The Truth Table of a RST Flipflop.</td>
<td>16</td>
</tr>
<tr>
<td>III</td>
<td>Discrete Values of Sine Function Generated by SIDAC-I Function Generators and FORTRAN Sine Function.</td>
<td>20</td>
</tr>
<tr>
<td>IV</td>
<td>The Experimental Results Obtained by Solving the Simple Harmonic Motion Equation by SIDAC-I Integrators.</td>
<td>27</td>
</tr>
<tr>
<td>V</td>
<td>The Truth Table of Logical Inputs of CINT Routine.</td>
<td>31</td>
</tr>
<tr>
<td>VI</td>
<td>Truth Table for NAND Gate.</td>
<td>34</td>
</tr>
<tr>
<td>VII</td>
<td>Truth Table for NOR Gate.</td>
<td>35</td>
</tr>
<tr>
<td>VIII</td>
<td>Truth Table for OR Gate.</td>
<td>37</td>
</tr>
<tr>
<td>IX</td>
<td>The Truth Table of Logical Inputs of the Track-Transfer.</td>
<td>48</td>
</tr>
</tbody>
</table>
INTRODUCTION

A. Church, an American mathematician, in 1931, stated a remarkable hypothesis, the so-called Church's Thesis which is equivalent to saying that all intuitively algorithmic functions can be computed by some Turing machines. (The definition of the Turing machine can be found in Appendix A.) This hypothesis together with the universal Turing machine theorem became the basic concept which leads to the development of this paper.

A universal Turing machine is a Turing machine which is constructed in such a way that it is capable of executing any algorithm. It is, in a certain sense, capable of doing the works of any Turing machine. The universal Turing machine is stated without proof as follows:

THEOREM:

There exists a Gödel index \( z \) such that for all \( x \) and \( y \),

\[
\phi_z(<x, y>) = \phi_x(y),
\]

where \( \phi_z \) and \( \phi_x \) are functions which are computable by a Turing machine. In other words, there is a Turing machine \( T_z \) with inputs \( x \) and \( y \) that simulates another Turing machine \( T_x \) with the input \( y \).
The word SIMULATION is yet to be defined. J. McLeod in his article, SIMULATION IS WHA-A-T? (10), defines SIMULATION as "... the act of representing some aspects of the real world by numbers or symbols which may be easily manipulated to facilitate their study." Specifically, the simulation we are dealing with in this paper can be defined as an act of representing the behavior of a Turing machine by another Turing machine.

In principle a general purpose digital computer is a special model of a Turing machine, a universal Turing machine. Thus, by the universal Turing machine theorem stated above, a general purpose digital computer has the capability of simulating some other machines. We are particularly interested in this paper in employing a general purpose digital computer to simulate a hybrid computer. Generally, a hybrid computer is recognized as a system which consists of both an analog computer and a high speed digital computer, with a suitable interface system for interconnection and control. However, a system which combines an analog computer and an assemblage of a patchboard-connected logical components is also referred to as a hybrid computer. This paper is concerned with the writing of a simulator program named SIDAC-I (SImulated Digital-Analog Computer) which will be used to make a digital computer (the Oregon State University IBM 1620) appear like a hybrid computer to the user.
Since the first attempt by Selfridge (14) in 1955, the extensive research in the field of digital simulation has been done. Simulators like DAS (3), PACTOLUS (2), and MIDAS (13), are fairly well known and well accepted as the powerful tools for analog simulation. The interconnection languages and the computational routines have been improved tremendously. PACTOLUS, an on-line control digital analog simulator recently developed by Brennan of International Business Machines Corporation, permits more "hand on" control of the program than most of the existing simulators. SIDAC-I was designed to have as many of the best features of these simulators as possible. The reader is reminded that SIDAC-I was developed for a comparatively small computer with medium speed; thus, several features which might have been incorporated with a larger and faster computer had to be eliminated. However, the main concern in the development of this program was the demonstration of the possibility of writing analog and hybrid simulation programs in the form of digital computer programs. Thus, a SIDAC-I coding program will have a form of "mixed" source program. A "mixed" source program refers to a source program which consists of both the patching instructions of SIDAC-I elements and the regular digital computer language instructions. As the name suggests, SIDAC-I is the first in the series of the SIDAC project. The SIDAC-I controller scheme and its functional routines are still awaiting for further improvement.
THE GENERAL DESCRIPTION OF THE SIDAC-I SIMULATOR

SIDAC-I was developed for the IBM 1620 computer with the Card Read-Punch. The IBM 1627 Plotter is optional. The program is subdivided into three distinct sections:

1. Main Control; this is the simulation of the interface system for interconnection and control.

2. The Simulated Patchbay; this is the simulation of the assemblage of a patchboard.

3. The library of Subroutines; this is the simulation of the standard analog computer elements plus some of the logical components such as flip-flops, NAND gates, etc. These are the pre-defined functional blocks which the programmer may use to build his system. Appendix B contains the complete list of the available elements and symbols.

The first two portions of the program are considered to be the execution programs. The simulated patchbay defines how the pre-defined functional blocks are to be interconnected. The control section determines how the run is to be made and how the input and output information is to be handled. This control section controls also the integration period, the total running time, and finally the headings. The information relating to the execution portions is entered into the program via punched-card formats. After the structure of
the program is specified the parameter and initial condition values for each block used need to be entered into the program. This can be done by reading the data cards through the card reader. SIDAC-I has flexibility in changing the control or parameter data for a given system structure so that several runs can be made without changing the patching instructions. This is the property that is commonly found in most of the digital simulators. However, there is one thing SIDAC-I has that is lacking in most simulators; it is the ability of accepting an additional running time. In other words, SIDAC-I is capable of continuing the computation for some additional time after it has stopped because of the previously specified final time. This is very useful when the programmer discovers that the final time he has specified is too short and that he has to have a longer running time in order to obtain a reasonable solution. Without going back to start all over at the beginning a SIDAC-I programmer may add as much additional computation time as he needs as often as he desires.

SIDAC-I is capable of solving problems described in terms of ordinary differential equations and at the same time performing the logical operations. It might be described as a block-oriented interpretive program, just as in analog simulation the programmer will draw a block-diagram to represent his system. For a hybrid-computer-type problem the program may be written with the analog (differential equation) portion in the form of block-oriented language
while the logical portion may be written in the block form and/or in
FORTRAN II language. The "patching" of the analog portion is per-
duced by a sequence of connection statements. The typical connec-
tion statement or patching instruction might take the form

\[ \text{DY} = \text{SINT}(\text{A5} \times \text{PL} / 5.0 + \text{G9}) \]  

(2-1)

which means connect the product of output A5 and output PL divided
by a constant 5.0 plus the output G9 to the input of an integrator;
DY is assigned as the name of the output of this integrator.

The SIDAC-I programmer does not have to worry about the
scaling problem as he does in the case of an analog simulation since
the scaling is automatic through the use of digital-floating-point
arithmetic. However, he must define parameters for each block as
well as the initial condition values of the integrators and the run in-
formation. Just like most of the digital analog simulators, SIDAC-I
has some advantages over the real hybrid computer systems and
real analog computer systems. For example, SIDAC-I gives much
more precision than can be achieved with analog equipment. A
SIDAC-I programmer is not required to have a knowledge of elec-
tronics. Finally the prewired system is not complicated because
there is no physical lead interconnection required. SIDAC-I also
has some disadvantages. The most obvious one is that it is not pos-
sible to change parameters quickly and easily. Even though the
simulator is capable of responding to interrupted signals for changing the initial condition and/or parameter values at any instant, the man-machine rapport is still rather poor.

SIDAC-I has two versions; the plot-version and no-plot-version. The difference between the two is that the plot-version is capable of plotting output data on a IBM 1627 plotter while the no-plot-version does not do this. The difference in the structure of the two simulators is in the main control sections. The rest of the simulators are identically the same except that the plot-version has a subroutine PLOTER* in addition to the other SIDAC-I elements while the no-plot-version has no such subroutine.

The Technique of Simulation

SIDAC-I was written in FORTRAN II language. By taking advantage of the outstanding feature of the FUNCTION and SUBROUTINE subprograms which the FORTRAN language has, a variety of special techniques regarding the writing of each of the SIDAC-I components, both analog and logical elements, can be developed. The essential fact is that both the FUNCTION and SUBROUTINE subprograms can be completely independent of the main program and yet the communication between any of the subprograms and the main program can be

*PLOTER is the name of the subroutine and it has only one "t". The reader must not confuse with the word "plotter" which has two t's.
set up quite easily. For example, an analog computer element, the
POSITIVE CLIPPER for instance, may be represented by a FUNC-
TION subprogram which is written in such a way that the output of
this FUNCTION subprogram is zero if the input signal is some posi-
tive value while it is identically equal to the input signal if the input
is negative. Each of the subprograms is assigned a name which the
computer can recognize distinctly. In order to make use of one of
these subprograms, it is only necessary to write its name and to
follow it with an expression enclosed in parenthesis. This expres-
sion can be the representation of inputs as well as the representation
of logical signal to that particular element. With the name assigned,
the element can be called for anytime it is needed in the program.
Furthermore, a main program may be written to function as a con-
trol center for communication among the simulated analog computer
elements, the logical elements, the simulated patchbay, and the
digital computer itself.

Since all the components are independent of each other the user
has to read into the computer only the subprograms which represent
the elements needed in solving a certain problem. This is obviously
a core storage saving type simulator and it is thus very feasibly used
on a small digital computer which has no disk file and has a small
size memory. This technique offers an advantage in another impor-
tant aspect, that is, at the end of each iteration period the output of
each element is assigned to a location in the memory from which it can be called by an arbitrary name or address. This provides the capability of linkage between the analog portion and the logical program. In other words, this technique makes it possible to have one or more FORTRAN statements appear anywhere in the program. This is in fact the essential property that SIDAC-I has in order to make it possible to program a simulation problem in the form of digital computer program.
THE DESCRIPTION OF THE SIDAC-I ELEMENTS

The description of the components which are available in SIDAC-I are presented in alphabetical order. The number of individual components is varied, subject to the restriction that the total number of components in any simulation be limited to an arbitrary number. This number is determined by the core storage required by a particular component and the availability of the core memory of the computer. The block diagram together with the source program of each SIDAC-I component can be found in Appendix D.

Automatic Stop

This component serves as a breaker for a simulation system. It accepts two input signals X1 and X2, respectively. X1 can be any variable of the program while X2 is normally the upperbound of X1. During each time increment X1 and X2 are brought together and compared. The computer is caused to halt as soon as the magnitude of X1 becomes greater than the magnitude of X2.

AND Gate

This routine is designed to be used as a Boolean binary operator AND. It is sometimes referred to as logical addition by logicians or as conjunction by mathematicians.
Table I. Truth Table for AND Gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A · B · C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table I is a truth table for the arguments A, B, and C. Each SIDAC-I AND gate is capable of accepting a maximum of five arguments. Thus the maximum possible number of combinations of zeros and ones for the various arguments is $2^5$. Note that an argument whose magnitude is less than zero is treated as if it were zero while an argument whose magnitude is greater than one is treated as if it were one. The output of an AND gate is either zero or one.

**Bang-Bang**

This is a simulation of an ideal analog switch. The output of this element switches between $+P$ and $-P$. The state of each elementary switch is determined by the input signal. The static
characteristics of this ideal switch are shown in Figure 1.

![Figure 1](image)

**Figure 1.** Transfer characteristics of a Bang-Bang.

The Bang-Bang yields an output signal which may be described by equation (3-1).

\[ X_2 = P \text{sgn} X_1 \quad (3-1) \]

**Clock**

This routine serves as a timing device to keep track of the time, or generally, to keep track of the independent variable. The output is a sequence of numbers which are some integral multiples of the current values of the independent variable. If the argument of this element is assigned as a one the output is identically equal to the independent variable. This function is always present in the program.
Comparater

This element is very useful when logical operation is required in the system. It compares the two input signals and yields an output whose value is either zero or one.

![Diagram of comparater transfer characteristics]

Figure 2. Transfer characteristics of a comparater.

Counter

The SIDAC-I counter has the characteristics of a zero crossing counter. It counts the number of times that the input signal is greater than zero. The period of the output is determined by the second input P. Note that the maximum value of the counter output is equal to the modulo the integral part of the second input, i.e., if

\[ P = 8.731 \]  
\[ \text{modulo} = 8.0. \]
Dead Space

This is the simulation of a nonlinear element which is commonly found in various physical systems. The dead zone is bounded by the parameters UB and LB which stand for upper bound and lower bound respectively. This type of characteristic is also referred to as a threshold nonlinearity. Its transfer characteristics are shown in Figure 4. The threshold gap has a length of UB + LB.
Delay

In a physical system a delay line is designed to delay the arrival of an electrical or acoustical signal at a given point by a predetermined period of time. In a simulation system the delay element was written to have a delay characteristic which is independent of the bandwidth or cutoff frequency. The SIDAC-I Delay component is designed to delay the arrival of a signal by one-half of the simulated time (one-half of an integration period). Delays of the order of some integral multiple of an integration period can be achieved with such a Delay element if a series of them are connected together.

This component can be applied for a variety of purposes such as pulse formation and pulse discrimination. The programmer will find it very helpful in synchronizing logical signals.

Figure 4. The transfer characteristics of the Dead-Space (threshold nonlinearity).
Flipflop

Because the hybrid simulation is dealing partly with the logical operation, elements for storing information are necessary. Flipflop elements can store binary information as either a one or a zero. A practical flipflop element can exhibit and maintain a given binary state. The change from one state to another is usually caused by an appropriate input pulse. There are four types of flipflops which are commonly used: they are the single-input flipflop, RS flipflop, JK flipflop, and RST flipflop. The major difference among these four types is the number and effect of the inputs in switching the binary state of the flipflop. SIDAC-I flipflops are RST flipflops. The truth table is shown in Table II with three inputs, R, S, and T.

Table II. The Truth Table of a RST Flipflop

<table>
<thead>
<tr>
<th>$R(t_0)$</th>
<th>$S(t_0)$</th>
<th>$T(t_0)$</th>
<th>$A(t_0)$</th>
<th>$A(t_0+t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
The input $R$ resets the Flipflop to the zero state; input $S$ sets it to the one state; input $T$ complements the state of the Flipflop. Note from Table II that no two inputs can occur simultaneously. However, it is possible that a SIDAC-I Flipflop will accept two or three inputs whose magnitudes are simultaneously equal to one. In any event, the input $T$ is designed to have precedence over both $S$ and $R$ while $S$ has precedence over $R$. If the input $T$ is tested to be one the inputs $S$ and $R$ are ignored. If the input $T$ is zero and the input $S$ is one then the input $R$ is ignored. Note also that any input greater than zero is treated as one while the negative input is treated as zero.

**Function Generator**

SIDAC-I is capable of generating any function with one variable, i.e., $X_2 = F(x_1)$, by employing the function generator subroutine. The precision of the result depends on the number of coordinate pairs which are available. Functions are approximated through the use of either linear interpolation or second-order interpolation.

(a). Linear Interpolation (6, p. 33-43): This method of linear approximation employs the concept of the first divided difference of $f(x)$. The first divided difference of $f(x)$ relative to $x_0$ and $x_1$ can be expressed in the form

$$f(x_0, x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$  \hspace{1cm} (3-4)
It is clear that $f(x_1, x_0) \approx f(x_0, x_1)$.

Thus the linear approximation may be expressed in the form

$$f(x_0, x) \approx f(x_0, x_1), \quad (3-5)$$

which is the same as to say

$$\frac{f(x_0) - f(x)}{x_0 - x} \approx \frac{f(x_0) - f(x_1)}{x_0 - x_1}, \quad (3-6)$$

which leads to the interpolation formula

$$f(x) = f(x_0) + \frac{x - x_0}{x_1 - x_0} [f(x_1) - f(x_0)]. \quad (3-7)$$

It may be noted that to generate a function whose independent variable lies between any two points $x_0$ and $x_1$ involves knowing two ordinates $f(x_0)$ and $f(x_1)$. Thus knowing the point which is below the lowest anticipated value of the independent variable of the function is necessary.

(b). Second-Order Interpolation: If the accuracy afforded by the previous method is inadequate a more accurate approximation may be used. The SIDAC-I function generator provides a second interpolation which is based on the supposition that $f(x)$ may be approximated by a polynomial of second degree near the abscissa of the interpolation. This is equivalent to assuming that, within a certain
prescribed tolerance, the first divided difference \( f(x, x_0) \) is a linear function of \( x \) for fixed \( x_0 \) or, equivalently, that the second divided difference \( f(x, x_0, x_1) \) is constant. The hypothesis

\[
f(x, x_0, x_1) \approx f(x_2, x_0, x_1) \approx f(x_0, x_1, x_2)
\]

then takes the form

\[
\frac{f(x, x_0) - f(x_0, x_1)}{x - x_1} \approx f(x_0, x_1, x_2)
\]

or, after another reduction,

\[
f(x) \approx f(x_0) + (x - x_0)f(x_0, x_1) + (x - x_0)(x - x_1)f(x_0, x_1, x_2).
\]

Substitute equations (3-4) and (3-8) in (3-9), after simplification,

\[
f(x) \approx f(x_0) + \frac{x - x_0}{x_1 - x_0} [f(x_1) - f(x_0)]
\]

\[
+ \frac{(x - x_0)(x - x_1)}{x_2 - x_0} \left[ \frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0} \right]
\]

These two interpolations are available in a SIDAC-I function generator. To choose one or the other method the user will only have to put a designator showing the order of interpolation in the patching instruction. Even though the second-order interpolation requires longer computation time the payoff given in terms of
precision is tremendous. Columns 2 and 3 in Table III shows the
discrete values of one full cycle sine wave which were generated by
these two interpolations (the list of instructions with data can be
found in Appendix C). Columns 4 and 5 are the relative errors of the
data in columns 2 and 3 with respect to the values in column 1 which
were obtained from the standard FORTRAN Sine function.

Table III. Discrete Values of Sine Function Generated by SIDAC-I
Function Generators and FORTRAN Sine Function.

<table>
<thead>
<tr>
<th>FORTRAN Sine</th>
<th>Linear Interpolation</th>
<th>Second-Order Interpolation</th>
<th>Error Linear Interpolation</th>
<th>Error Second-Order Interpolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.12533</td>
<td>0.12325</td>
<td>0.12913</td>
<td>0.00208</td>
<td>0.00380</td>
</tr>
<tr>
<td>0.42578</td>
<td>0.41297</td>
<td>0.43267</td>
<td>0.01280</td>
<td>0.00689</td>
</tr>
<tr>
<td>0.68454</td>
<td>0.65765</td>
<td>0.69249</td>
<td>0.02688</td>
<td>0.00795</td>
</tr>
<tr>
<td>0.80901</td>
<td>0.80446</td>
<td>0.80910</td>
<td>0.00454</td>
<td>0.00009</td>
</tr>
<tr>
<td>0.95105</td>
<td>0.85605</td>
<td>0.93147</td>
<td>0.09499</td>
<td>0.01958</td>
</tr>
<tr>
<td>1.00000</td>
<td>0.89532</td>
<td>0.97789</td>
<td>0.10467</td>
<td>0.02210</td>
</tr>
<tr>
<td>0.42577</td>
<td>0.44997</td>
<td>0.44735</td>
<td>0.02420</td>
<td>0.02157</td>
</tr>
<tr>
<td>0.18737</td>
<td>0.19284</td>
<td>0.19059</td>
<td>0.00547</td>
<td>0.00322</td>
</tr>
<tr>
<td>-0.63743</td>
<td>-0.59630</td>
<td>-0.66484</td>
<td>0.04112</td>
<td>0.02741</td>
</tr>
<tr>
<td>-0.92978</td>
<td>-0.85310</td>
<td>-0.94358</td>
<td>-0.07667</td>
<td>0.01380</td>
</tr>
<tr>
<td>-0.99802</td>
<td>-0.98573</td>
<td>-0.99455</td>
<td>0.01229</td>
<td>0.00347</td>
</tr>
<tr>
<td>-0.84432</td>
<td>-0.74922</td>
<td>-0.80671</td>
<td>0.09510</td>
<td>0.03760</td>
</tr>
</tbody>
</table>

Integrator

One of the more significant features of hybrid or analog simu-
lators is their ability to implement accurate integration. The
trapezoidal rule integration routine was found to be the simplest method but it is also the most undesirable one. The shortcomings of this method are the accumulation of error and inaccurate integration. The degree of accuracy is not satisfactory unless the time interval is chosen to be very small. The most popular integration routine which has been used for some large size and higher speed computers is the Predictor-Corrector integration routine (13). The step size in this type of integration routine adjusts itself to meet a certain error criterion, a factor which allows it to take large steps for those portions of the solution where the change of variables are not rapid and small steps for those portions when one or more variables are changing at rapid rates. The integrations in SIDAC-I are performed by a fixed-step, second-order, Runge-Kutta integration routine. Unlike the variable-step Predictor-Collector integration formulas, the inordinately long solution times are the cost of accurate results in this type of numerical integration. The theory of the second-order Runge-Kutta integration routine in SIDAC-I is discussed below.

Because SIDAC-I is inherently a sequentially operated simulator program, the computation of derivative and integration processes have to be done separately within two different time intervals. A tremendous error will be introduced otherwise. This is due to the fact that the first integrator processes the integration one time
increment ahead of the other integrators in the same feedback loop.

In other words, during an integration period i a jth integrator whose output is the input to a (j + 1)th integrator updates its output to $Y_{ji}$. $Y_{ji}$ automatically becomes the current derivative for the integrator $j + 1$. This is obviously an error because the derivative at integration period i is supposed to be $Y_{i-1}^j$ rather than $Y_{i}^j$. One way to avoid this error is to divide each integration period into four sub-intervals. Figure 5 shows the control pulses during one integration period.

![Diagram](image-url)

Figure 5. The control pulse waveforms in one integration period.

The calculation of derivatives and integrations are done separately and they are controlled by the control signal ISKIP. The integration routine computes the derivatives when ISKIP is set to the one state and integrates when ISKIP is reset to zero. The integration
algorithm can be stated as follows:

(1). Let an index $i$ designate the $i$th integration period. Start with $i = 1$. This is the state where the initial condition values of all integrators are to be defined. Calculate the first derivative

$$y'_i = f(x_i, y_i).$$

(2). Set the index $i = i + 1$. The $i$th integration period is divided into four subintervals designated as $i_1, i_2, i_3,$ and $i_4$.

(3). During the first subinterval the calculation of the first portion of integration of all integrators takes place. This is analogous to locating a point $P$ in Figure 6. It may be noted that point $P$ is located with the ordinate erected at $x_i + \frac{1}{2}h$ where $h$ is the spacing,

$$h = x_i - x_{i-1}.$$  \hspace{1cm} (3-12)

The numerical integration is performed by the equation (9, p. 320-324);

$$y_i - \frac{1}{2} = y_i - 1 + \frac{1}{2}hy'_i - 1,$$  \hspace{1cm} (3-13)

where

$$y'_i - 1 = f(x_i - 1, y_i - 1).$$  \hspace{1cm} (3-14)

It may be pointed out at this point that the value $y'_{i-1}$ was computed in state (1) when the index $i = 1$ and in state (6) when $i > 1$. 
Figure 6. The geometrical representation of the second-order Runge-Kutta numerical integration method.

(4). During the second subinterval \( i_2 \) the derivative of \( Y_{i}^{'} - \frac{1}{2} \) is calculated,

\[
Y_{i}^{'} - \frac{1}{2} = f(x_{i} - 1 + \frac{1}{2}h, y_{i} - 1 + \frac{1}{2}h y_{i}^{'}, y_{i - 1}).
\]  

(3-15)

This process is analogous to drawing a line \( L_2 \) through the point \( P \) with the slope \( y_{i}^{'} - \frac{1}{2} \).

(5). During the third subinterval \( i_3 \) the integrator starts the second integration process with the derivative which was previously computed in state (4). The corresponding equation is

\[
y_{i} = y_{i} - 1 + h y_{i}^{'} - \frac{1}{2}.
\]  

(3-16)

This process is analogous to drawing a line \( L_0 \) parallel to line \( L_2 \) passing through the point \( (x_{i} - 1, y_{i} - 1) \). The line \( L_0 \) intercepts
the line $[(x_1, 0), (x_1, y_1)]$ at point $(x_1, y_1)$. (See Figure 6).

(6). During the fourth subinterval $i_4$ the derivative of $y_1'$ is calculated,

$$y_1' = f(x_1, y_1)$$  \hspace{1cm} (3-17)

$y_1'$ is then stored as an integrand for the $i + 1$ integration period. This is equivalent to drawing a line $L_{11}$ through the point $(x_1, y_1)$ with the slope $y_1'$.

(7). A test is made to see if the time is equal or greater than the specified final time; if it is not, then go back to continue from state (2); if it is then the process is terminated.

It may be noted that the integrators may integrate with respect to variables other than time.

The performance of this integration routine can be tested to see the degree of accuracy relative to the time increment by the method which is present as follows. This method will also show the behavior of the accumulation error.

The method involves solving a harmonic motion differential equation

$$\frac{d^2X}{dt^2} + X = 0$$  \hspace{1cm} (3-18)

which has the solution

$$X = A \cos (t + \phi)$$  \hspace{1cm} (3-19)
and its derivative

\[ \frac{dX}{dt} = B \sin (t + \phi) \quad (3-20) \]

Thus, with some proper initial condition values the integrator can be used to generate \( \cos (t + \phi) \) and \( \sin (t + \phi) \); theoretically,

\[ \sin^2(t + \phi) + \cos^2(t + \phi) = 1 \quad (3-21) \]

Table IV shows the discrete values of equations (3-19), (3-20), and (3-21) which were obtained by the SIDAC-I integrators. The errors were obtained by subtracting the experimental result of equation (3-21) by a value of one. The list of SIDAC-I coding instructions with data can be found in Appendix C.

Quite often, a programmer will find himself wanting an integrator which is able to do more than just the ordinary integration. In doing most of the interactive computation it is very desirable to have decision-making elements and digital logic function to switch analog equipment. A mode controlled integrator is a special kind of integrator which has a track-hold memory unit built in. SIDAC-I has both simple integrators and mode controlled integrators.

(a). Simple integrators: This type of integrator can perform integrations only. Before every computer run the correct initial value of the integrator output must be specified. This process is analogous to charging the integrating capacitor \( C \) in each electronic
Table IV. The Experimental Results Obtained by Solving the Simple Harmonic Motion Equation by SIDAC-I Integrators.

<table>
<thead>
<tr>
<th>The Independent Variable t</th>
<th>$\sin(t + \phi)$</th>
<th>$\cos(t + \phi)$</th>
<th>$\sin^2(t + \phi)$</th>
<th>$\cos^2(t + \phi)$</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.00000</td>
<td>1.00000</td>
<td>1.00000</td>
<td></td>
<td>0.00000</td>
</tr>
<tr>
<td>2.000</td>
<td>0.90813</td>
<td>-0.41927</td>
<td>1.00050</td>
<td></td>
<td>0.00050</td>
</tr>
<tr>
<td>4.000</td>
<td>-0.76151</td>
<td>-0.64892</td>
<td>1.00100</td>
<td></td>
<td>0.00100</td>
</tr>
<tr>
<td>6.000</td>
<td>-0.27003</td>
<td>0.96363</td>
<td>1.00150</td>
<td></td>
<td>0.00150</td>
</tr>
<tr>
<td>8.000</td>
<td>0.98832</td>
<td>-0.15879</td>
<td>1.00200</td>
<td></td>
<td>0.00200</td>
</tr>
<tr>
<td>10.000</td>
<td>-0.55858</td>
<td>-0.83095</td>
<td>1.00250</td>
<td></td>
<td>0.00250</td>
</tr>
<tr>
<td>12.00</td>
<td>-0.52042</td>
<td>0.85566</td>
<td>1.00300</td>
<td></td>
<td>0.00300</td>
</tr>
<tr>
<td>14.000</td>
<td>0.99526</td>
<td>0.11385</td>
<td>1.00350</td>
<td></td>
<td>0.00350</td>
</tr>
<tr>
<td>16.000</td>
<td>-0.31388</td>
<td>-0.95156</td>
<td>1.00400</td>
<td></td>
<td>0.00400</td>
</tr>
<tr>
<td>18.000</td>
<td>-0.73255</td>
<td>-0.68401</td>
<td>1.00451</td>
<td></td>
<td>0.00451</td>
</tr>
<tr>
<td>20.000</td>
<td>0.92831</td>
<td>0.37846</td>
<td>1.00501</td>
<td></td>
<td>0.00501</td>
</tr>
</tbody>
</table>

a. The time increment = 0.1
Table IV. Continued.

The Independent Variable $t$  

<table>
<thead>
<tr>
<th>$t$</th>
<th>$\sin (t + \phi)$</th>
<th>$\cos (t + \phi)$</th>
<th>$\sin^2(t + \phi)$</th>
<th>$\cos^2(t + \phi)$</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.00000</td>
<td>1.00000</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>2.000</td>
<td>0.90928</td>
<td>-0.41617</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>4.000</td>
<td>-0.75684</td>
<td>-0.65359</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>6.000</td>
<td>-0.27933</td>
<td>0.96019</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>8.000</td>
<td>0.98934</td>
<td>-0.14562</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>10.000</td>
<td>-0.54415</td>
<td>-0.83898</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>12.000</td>
<td>-0.53641</td>
<td>0.84395</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>14.000</td>
<td>0.99064</td>
<td>0.13651</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>16.000</td>
<td>-0.28815</td>
<td>-0.95758</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>18.000</td>
<td>-0.75079</td>
<td>0.66053</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>20.000</td>
<td>0.91307</td>
<td>0.40078</td>
<td>1.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

b. The time increment = 0.01
integrating circuit to the correct initial value of the integrator output voltage. In the case of electronic integrators one of the common methods which specify the initial condition is by means of connecting a low-impedence d-c source to charge the integrator capacitors during the RESET mode. To start a computer run the reset relays of all integrators are de-energized by means of a Compute-Reset switch. In the case of SIDAC-I integrators the initial values are specified through data cards. The procedure of reading initial values will be discussed in detail in a latter section under the heading DATA DECK (p. 60).

(b). Mode controlled integrators: Figure 7 illustrates a practical mode controlled integrator for a slow electronic differential analyzer (8, p. 374-80). It may be noticed that there are two mode-control relays, i.e., the RESET mode relay and the HOLD mode relay which control three modes of operation, i.e., COMPUTE mode, RESET mode, and HOLD mode. The compute (operate) mode permits the normal integration, while the hold mode permits the integrator to hold the last computed output voltage. This is equivalent to having an analog memory in an integrator. Energizing both reset relay, \( K_r \), and the hold relay, \( K_h \), brings the integrator to the RESET mode. In the RESET mode the output of the integrator has the value of the initial condition (IC) input to the amplifier.

The application of the HOLD mode is for slow readout into
digital voltmeters, printers, etc., and for manual or automatic program changes. For slow iterative computation the HOLD mode provides analog storage. For slow repetitive operation the RESET mode is useful for resetting the computation periodically.

Figure 7. An electronic mode controlled integrator (8, p. 374).

To simulate this component, a SIDAC-I mode controlled integrator has two inputs P1 and P2 to control the mode of operation. The logical input P1 is the simulation of the HOLD relay while P2 is the simulation of the RESET relay. Input P1 is tested first and if found to be positive the integration is performed, otherwise, P2 is tested. Finding P2 non-positive brings the integrator to its HOLD
mode, otherwise the integrator is in the RESET mode. The RESET mode is also referred to as the TRACK mode since as long as the integrator is held in this mode, its output follows, or tracks, the value of the IC input. Table V shows the truth table of these two logical inputs and the corresponding integrator modes.

Table V. The Truth Table of Logical Inputs of the CINT Routine.

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>integrator mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>operate</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>operate</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>track (IC)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>hold</td>
</tr>
</tbody>
</table>

For SIDAC-I mode controlled integration the method of integration is the same as for simple integrators. The initial condition may be set at t = 0, by the value of the IC input of the Mode Controlled Integrator. It may be noted that the initial condition cannot be set by an entry in the initial condition cards. But it may be set by an appropriate entry in the parameter cards.

(c). Memory-pair operation: Various applications may arise when two or more mode controlled integrators are connected in cascade. Figure 8 illustrates an application of a memory-pair comprising two cascaded track-hold elements. To be used as a track-hold element, the logical input P1 of the mode controlled integrator
Figure 8. Illustration of a memory-pair operation (8, p. 398).
is set at zero while P2 is set either at one or zero depending on the mode of operation.

The control pulse P2 for the track-hold 2 are delayed so that the element tracks the hold output of track-hold 1 and stores or presents it while track-hold 1 is free to track again. The memory-pair output is a delayed sequence of clean sampled-data steps.

Limiter

The SIDAC-I limiter has the transfer characteristics shown in Figure 9. This is the simulation of saturation nonlinearity whose characteristics approximate the behavior of various amplifiers, valves, motors, etc.

![Figure 9. The transfer characteristic of SIDAC-I limiter.](image-url)
It should be noted that the breakpoints P1 and P2 are not fixed. The breakpoints may be set by an appropriate entry in the patching instruction. This flexibility makes it possible for the simple limiter to simulate important properties of dynamical systems. In practice, limiters can be combined to generate a large class of arbitrary functions.

**NAND Gate**

This is another basic logic element which is designed to perform elementary logic operations. The NAND operation, first introduced by Sheffer in 1913, is also known as Sheffer's stroke function. It is the complement of the logical-AND operation. Table VI is the truth table for the arguments A, B, and C. Each SIDAC-I NAND gate is capable of accepting a maximum of five arguments. An argument whose magnitude is less than zero is treated as zero. Any positive value argument is treated as one. The output of a NAND gate is either zero or one.

**Table VI. Truth Table for NAND Gate**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>(A \cdot B \cdot C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Negative Clipper

The SIDAC-I negative clipper accepts one input signal. It was designed to clip the negative portion of the input waveform.

NOR Gate

The NOR operation is also known as Peirce's function. A NOR B means the complement of A or B or both, that is, \(A + B\). It is the complement of the logical OR operation. Table VII shows the truth table of the SIDAC-I NOR gate. Each SIDAC-I NOR gate accepts up to five inputs. An argument whose magnitude is less than zero is treated as zero. Any positive value argument is treated as one. The output of a NOR gate is either a zero or a one.

Table VII. Truth Table for NOR Gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>(A + B + C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
**NOT Gate**

The logical NOT signifies the meaning of the word negation, or complement. The logical NOT proposition of a given proposition is true (or false) if that given proposition is false (or true). The SIDAC-I NOT gate simulates the above characteristic. The SIDAC-I NOT gate accepts one argument. The output of the gate is one for the positive portion of the input and zero for the negative portion of the input.

**Offset**

This element is capable of shifting a signal by a fixed amount specified in the patching instruction. The programmer will find it very helpful in scaling the x - y coordinates when the plotter is used.

**OR Gate**

The SIDAC-I OR gate has the characteristics of an inclusive OR. Table VIII shows the truth table of the OR gate.

Each SIDAC-I OR gate accepts a maximum of five arguments. An argument whose magnitude is less than zero is treated as zero. Any positive value argument is treated as one. The output of an OR gate is either a zero or a one.
Table VIII. Truth Table for OR Gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A + B + C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

PARAMETER

To demonstrate its operational flexibility any digital simulator will have to have a convenient means of changing the parameters and/or initial values both automatically and manually. The PARAMETER routine was designed for this purpose. The following are the two applications of the SIDAC-I PARAMETER:

1. It may be used to identify the variables in the simulation programs. Any name which appears in a SIDAC-I PARAMETER calling statement can be used freely as the name of a variable throughout the program. The numerical value in the PARAMETER cards (in the data deck) is the initial value of this variable.

2. It may be used to set the initial condition value for a mode controlled integrator. This is very useful when the IC terminal of a mode controlled integrator is connected to an AC source.
In either of the two cases the SIDAC-I PARAMETER has no other functions whatsoever after the initial condition values have been assigned to the variables which appear in the calling statement. Assigning the initial conditions to the parameters is done at time \( t = 0^+ \). This component is skipped throughout the entire computer computation for \( t > 0^+ \).

The SIDAC-I PARAMETER is capable of identifying a maximum of 20 parameters. Each PARAMETER calling statement is allowed to call from one to a maximum of five variables. The first argument has to be an integer \( N \), a designator, to indicate the number of parameters appearing in a single calling statement. The calling statement has the following format,

\[
\text{CALL PAR}(N, \text{P1}, \text{P2}, \text{P3}, \text{P4}, \text{P5}) \tag{3-22}
\]

where \( 1 \leq N \leq 5 \).

A maximum of four calling statements are allowed in a program. A dummy variable is needed in the place where the argument is not used in any of these calling statements.

**PLOTER**

When an IBM 1627 plotter is available SIDAC-I output information can be obtained in graphic form. The subroutine PLOTER is designed to cause the plotter to plot point by point from coordinates supplied by the arguments of the PLOTER calling statement. The
calling statement has the following format:

\[ \text{CALL PLOTER(X, Y)} \]  \hspace{1cm} (3-23)

X is designated as the variable representing the abscissa while Y is variable representing the ordinate. The grid size is fixed with the maximum of both X and Y at +100.0 and the minimum of both X and Y at -100.0

\[ (-100, 100) \]

\[ (-100, -100) \] \hspace{1cm} (100, -100)

Figure 10. The SIDAC-I plot frame with the fixed grid size.

The programmer has to scale the x - y coordinates properly when PLOTER is used. The scaling can be done via the use of the Offset routine together with some appropriate gain constants. As an illustration, let us consider the case where the predicted maximum x and y values are 2.5 and 100.0 respectively. The scaling procedure can be done by the following steps shown as follows:
The patching instructions which scale the x - y coordinates have the following forms:

\[ XAXIS = OFFSET(80.0 \times X, -100.0) \]  
\[ YAXIS = OFFSET(2.0 \times Y, -100.0). \]

![Diagram](image.png)

Figure 11. Offset elements and gain constants are used to assist the scaling of x - y coordinates.

Here we are dealing with four gain constants, i.e., \( XK, XP, YK, \) and \( YP \) which take the values of 80.0, -100.0, 2.0, and -100.0 respectively in the above example. These values can be calculated by the following general equations:

\[ XK = \frac{XMAX - XMIN}{XMAX_p - XMIN_p} \]  
\[ YK = \frac{YMAX - YMIN}{YMAX_p - YMIN_p} \]
\[ XP = XMIN - (XK \times XMIN_p) \]  
\[ YP = YMIN - (YK \times YMIN_p) \]  

\( XMIN_p \) and \( XMAX_p \), are the predicted minimum and maximum value of \( X \) while \( YMIN_p \) and \( YMAX_p \), are the predicted minimum and maximum value of \( Y \). \( XMAX \) and \( YMAX \) are both fixed at 100.0 while \( XMIN \) and \( YMIN \) are fixed at -100.0.

Equations (3-26), (3-27), (3-28) and (3-29) now can be written as

\[ XK = \frac{200.0}{XMAX_p - XMIN_p} \]  
\[ YK = \frac{200.0}{YMAX_p - YMIN_p} \]  
\[ XP = -100.0 - (XK \times XMIN_p) \]  
\[ YP = -100.0 - (YK \times YMIN_p) \]  

SIDAC-I is capable of executing three different problems in one loading. After the first program has been executed, regardless of whether or not the programmer is willing to start the new problem, the computer will type out the following message:

**TURN ON SENSE SWITCH 3 FOR NEW PLOT FRAME.**

The computer will then come to a halt. The Sense Switch 3 must be turned on if a new plot frame is to be drawn. Turning on Sense
Switch 3 on the console causes the pen to move to the up status and provides a margin of 2.5 inches between the two plot frames. In case the programmer wants the new set of data to be plotted on the old plot frame superimposing with the previous graph, he may leave Sense Switch 3 off and press the start key to continue the computation.

**Positive Clipper**

The SIDAC-I positive clipper accepts one input signal. It was designed to clip the positive portion of the input waveform.

**Pulse Generator**

The SIDAC-I pulse generator is capable of generating a series of pulses with a pulse width of one integration period. Choice of frequency and amplitude is left to the discretion of the programmer.

![Figure 12. Pulses generated by SIDAC-I pulse generator.](image)
Punch-output

The SIDAC-I output can be punched on a card by the PUNCH subroutine. The names of the variables for which values are to be punched are listed as arguments in the Punch calling statement. Each value is punched in floating point. The programmer will have to specify the punch interval in terms of an integral multiple of the integration period. Each value is punched out in the form of F-field format, F15.5, with the first-named variable always being the first on the card.

Each PUNCH statement is allowed to call from one to a maximum of five variables. A dummy constant is required in the place of the unused argument. It is recommended to assign a value of zero to these dummy constants in the floating-point format.

RANDOM Noise Generator

The most commonly used method of generating random numbers is the following (5, p. 388):

\[ X_{n+1} = \phi X_n \pmod{\text{word length of the machine}} \] (3-34)

Thus one uses one multiplication per number and takes the low-order digits of the product for the next number.

Another commonly used method is the one which was suggested
by von Neumann (11, p. 88). One takes an $n$ digit number, squares it and uses the middle $n$ digits as a random number. The latter is in turn squared, and the process continues.

It is interesting to note that in simulating a random noise generator on a digital computer it is less complicated to use the first method than the von Neumann method. Although the von Neumann scheme is as simple as the first method but obtaining the middle $n$ digits from a number requires extra computer time. On the contrary, obtaining the low-order digits of a number is quite simple because it can be done automatically by the computer. For the computer to do it automatically we mean the following:

The IBM 1620 processor at the Oregon State University Computer Center has the range of integers which may be optionally set by the programmer to a fixed number of either four or five as the "word size". The value of five is selected when the computer is doing the SIDAC-I computation. Thus the range of integers is from -99999 to +99999. But as far as doing the computation in the computer is concerned, all numbers are represented by strings of ten digits. Consequently, it does not matter how large a number is only its five low-order digits are taken as the result.

By taking advantage of this property of the IBM 1620 processor, the subroutine of the SIDAC-I RANDOM noise generator uses the following algorithm:
1. Set index \( n = 0 \).

Set \( \hat{\phi} = 237 \)

\( \text{IX} = 7243 \)

2. Set \( n = n + 1 \) and compute

\( \text{IX} = 237 \times \text{IX} \) \hspace{1cm} (3-35)

\( \text{IX} \) is the five low-order digits of the product which was obtained from equation (3-35).

3. In order to obtain the random numbers whose magnitudes are bounded between \( \pm 1.0 \), it is necessary to compute

\( X = \text{IX} \) and

The random number = \( X / 50000.0 - 1.0 \) \hspace{1cm} (3-36)

4. Stop if the time equals to the final time, otherwise go to 2.

It should be noted that \( \text{IX} \) represents the number in the fixed-point form while \( X \) represents the same number in floating-point form. The following is the sequence of the first ten random numbers which were generated by the SIDAC-I RANDOM noise generator:

\(-.66818, -.35866, .99758, .42646, -.92898, -.16826, .12238, 
-.99594, -.03778, -.95386, \ldots \)

Relay

An electronic relay is a device that permits switching control of one or more circuits and is accomplished by electrical variations in a usually non-interdependent control circuit. In the usual
connotation, a relay is an electromagnetic device in which control power is supplied to the coil of a relay whose mechanical motion is used to actuate contacts of other circuits. The SIDAC-I RELAY has the property of being able to transmit either of two input signals. The latching of either of the two inputs is controlled by the third input C. The pictorial representation of Relay is shown in Figure 13.

![Diagram of Relay](image)

C is the switching signal

Figure 13. The pictorial representation of SIDAC-I Relay.

**Sawtooth Generator**

The wave form generated by the SIDAC-I sawtooth generator has a shape which is similar to that of Figure 14. Choice of frequency and amplitude is left to the discretion of the programmer. The period is expressed in terms of integral multiples of the integration period. The sawtooth wave is being built up during each integration period with an increment of \( H/(NT - 1) \); \( H \) is the amplitude and \( NT \times \Delta t \) is the period.
\[ T = NT \Delta t \]
\[ \Delta H = H/(NT - 1) \]

Figure 14. The pictorial sawtooth wave of SIDAC-I sawtooth generator.

Square-wave Generator

Just as in the case of the Sawtooth generator the choice of frequency and amplitude of the square wave is left entirely to the discretion of the programmer. Figure 15 shows a typical square wave generated by the SIDAC-I Square-Wave Generator.

Figure 15. The typical square wave generated by SIDAC-I Square-Wave Generator.
Track-Transfer

This element is capable of tracking, transferring, and storing the input signals. There are two logical signals to control the modes of operation. With the proper setup of the control signal the SIDAC-I Track-Transfer can store the input in its virtual memory. This stored information may be transferred to its output terminals by control signals. The output is identical to the input when this element is in the "track" mode. Table IX shows the truth table of logical inputs A and B and the corresponding modes of operation.

Table IX. The Truth Table of Logical Inputs of the Track-Transfer.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Modes of Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( E_0 = E_m ) (transfer) or (hold)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( E_0 = X_2 ) (track)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( E_0 = E_m, E_m = X_1 ) (transfer, store)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>( E_0 = X_2, E_m = X_1 ) (track, store)</td>
</tr>
</tbody>
</table>

X1 and X2 are the two inputs while \( E_m \) is the value which was previously stored in the memory of Track-Transfer. \( E_0 \) is the output of the element. It should be noted that this element has characteristics which are similar to a mode controlled integrator when it operates at track-hold modes. The SIDAC-I Track-Transfer is useful in generating some logical control wave forms. Various types
of wave shapes can be generated when several of these Track-Transfer elements are connected in series. Figure 16 illustrates an example of a wave shaping element comprising two cascaded SIDAC-I Track-Transfers.

**Type-Output**

This element gives the typewriter output. This is for floating-point data only. The field specification of F15.5 is used. The names of the variables for which values are to be typed are listed as arguments in the Type calling statement. The first-named variable is always the first on the output sheet. The type interval is specified in the time control card. A TYOUT statement is allowed to call from one to a maximum of five variables. A dummy floating-point constant is required in the place of an unused argument.

**Zero-Order Hold**

This element accepts a sampling-switch signal with the sampling interval as input. The sampling process has a uniform rate, that is, a fictitious sampler closes for a very short interval at discrete periodic times \( t=0, T, 2T, 3T, \ldots, nT \ldots \). A continuous input to a SIDAC-I zero-order hold is sampled and the values of this input at the sampling instant are held constant until the next sampling instant. The output of the SIDAC-I zero-order hold is a step
Figure 16. Two cascade SIDAC-I Track-Transfers forming a wave shaping element.
approximation of the continuous input signal. Increasing the sampling rate of the sampler (increasing the frequency of the sampling input) tends to improve the approximation of the continuous signal. The idealized input and output waveforms of a SIDAC-I Zero-Order Hold element are shown in Figure 17b and Figure 17c, respectively. The reader is urged to remember that the sampling time must be short with respect to the time of any variation in the input signal, but it must not be shorter than the integration period. In other words, a complete sampling interval must not be less than one integration period.
Figure 17. SIDAC-I Zero-Order Hold operation in the time domain.
(a) Block diagram of the system.
(b) Input signal and sampled signal.
(c) Input signal and output waveform of zero-order hold.
SIDAC-I COMPUTER CODING

SIDAC-I can be used to solve ordinary analog simulation problems as well as hybrid simulation problems. In any case, the combinations of differential, algebraic, and logical equations, expressed in block-oriented form, may easily be coded for digital solution. All coding must be written in SIDAC-I simulation language which consists of the SIDAC-I block-oriented language and FORTRAN II. The preparation of a program consists of the following steps:

1. Write the differential equations for the system to be studied with the highest derivative on the left as is done in analog programming.

2. Draw an analog computer block diagram for the equations. Assign to the output of each component a name which is acceptable by the FORTRAN compiler. Choice of variable names for the SIDAC-I components is left to the discretion of the programmer, except for the restriction that they must be floating-point variables. The name of a floating-point variable has one to six letters or digits. The first of which is a letter, but not I, J, K, L, M, or N (9, p. 8).

Program Structure and Its Composition

Program Structure. It was mentioned previously that the simulation program entered the computer as a SUBROUTINE
subprogram. The main-control of the SIDAC-I simulator recognizes this particular subprogram by the name SIDACn, where \( n \) is an integer 1, 2, or 3. This integer \( n \) plays an important role in identifying the program which it represents. Since SIDAC-I is capable of executing three consecutive programs in one loading, it must be able to address these three programs separately. Thus SUBROUTINE SIDAC1 was designed to be the address of the first program. SUBROUTINE SIDAC2 and SUBROUTINE SIDAC3 are the addresses of the second and the third program respectively. The simulation program will, in general, be of the form,

```
SUBROUTINE SIDACn(I, A, B, C, D)

SIMULATION INSTRUCTIONS

RETURN

END
```

There may be as many simulation instruction as are required by the program as long as they do not exceed the capacity of the core memory of the computer. *

* The Oregon State University IBM 1620, at the present time, has a 40K core memory. There will be an additional unit of 20K in the near future.
There must be five arguments appearing in the SUBROUTINE SIDACn statement. The first argument must always be a fixed-point variable while the other four arguments are always floating-point variables. What do these arguments represent? The answer to this question is given below.

We had mentioned previously that the computations specified by the simulation configurations must be performed for each time cycle in some particular order. The general rule says that the computation for any block should not be attempted until all its inputs have been computed during that iteration. It is quite obvious that a system which consists of one or more feedback loops cannot follow this rule at the start point or when $t = 0^+$ because the computation for the first block of a feedback loop is attempted while its inputs have not yet been defined. The first integrator in Example 2, p. 83, is a good example. One can see, by considering its patching statement,

$$ SY = CINT(0.0, A, 1.0, 1.0-2.0\times Z\times SY - Y) \quad (4-1) $$

that the inputs $SY$ and $Y$ have not yet been defined at time $t = 0^+$. This is the point where the four floating-point arguments in the SUBROUTINE SIDACn statement start to play their roles. Thus the SIDAC-1 programmer may list all the initiated undefined variable names as the arguments in the SUBROUTINE SIDACn statement.
The problem of the "initiated undefined inputs" is solved as long as the first component of a feedback loop is an integrator. The reason behind this restriction is as follows:

During the initializing state the computer will do two things. First, it will initialize the integration algorithm. Secondly, it will define the inputs of every component. Thus the inputs to an integrator has no effect on the output of this integrator at \( t = 0^+ \) and the integrator can care less what the values of its inputs are. The problem still remains for the computer will execute the expression inside the parenthesis of the patching statement. The way SIDAC-I was designed to handle this difficulty is by having all the arguments in the SUBROUTINE SIDACn statement predefined in the MAIN CONTROL program, i.e., assign to each argument an arbitrary constant. Thus all but the fixed-point variable argument were assigned to have the value of 1.0. After having done this, any variable name which is listed in the place of an argument of this SUBROUTINE SIDACn statement is caused to have a value of 1.0. The initially undefined inputs are now defined and the expression in the patching statement can be executed properly. Emphasis is made at this point that the process of assigning a dummy constant to an undefined input can be done at \( t = 0 \) to the integrators only.

While the four floating-point variables arguments were assigned to have the values of 1.0 the fixed-point variable argument
was assigned to have the value of the ISKIP control signal. Thus having these arguments is analogous to having one pulse generator and four d-c voltage sources. Also, assigning a variable name to an argument is analogous to having a cord connected to a voltage source.

The reader is reminded that the ISKIP controlled signal plays an important role as the computation timing signal. When the ISKIP has the value of one, none but the derivative inputs to the integrators are computed. It is strongly recommended that use be made of this signal in hybrid simulation programs when the logical operations are to be skipped during the time that all the integrators are performing their derivative computation, i.e., when ISKIP = 1, for the sake of saving the computation time. An example of using this signal in this type of operation can be found in Example 2, p. 83, where this signal was assigned to have a name I. In brief, the first argument in the SUBROUTINE SIDACn statement may be used as the timing signal in some hybrid simulation problems. The initially undefined variable names are to be listed in this statement as the other four arguments. A dummy variable is required in the place of an unused argument.

There is no definite way to pin down the maximum number of the components that may be used. The programmer will have to be careful to not have the size of the program exceed the core storage that is available. It is his duty to keep track of the availability of
the core storage by adding up the lengths of all the components that appear in a program. The information regarding core storage requirements of each component can be found in Appendix B.

It is feasible to have a program written with a small number of instruction statements for the sake of storage saving, but it is not advisable to have the program reduced to an oversimplified form. The programmer may decide to add more blocks or to make changes in the existing program. Thus, it is practical to have the input of any component comprised of a small group of operands. As it has been noted that any operation may be performed at the input of any component, but this should not be abused for the sake of elimination of program redundancy. The user is reminded that any FORTRAN II statement may appear anywhere in the program. He might employ this property to aid him in obtaining a better program structure. If the FORTRAN II statements are part of the logical operation algorithm they are commonly grouped outside the patching statements.

**Patching Format.** All patching instructions must be described in 80 column IBM cards. The statement itself is punched in columns 7 to 72. If more than one card is required for a statement, the cards after the first must be punched with some non-zero character in column 6. A statement may not occupy more than nine cards. Components are patched by listing the variable name, an equal sign, the name of the function being represented, and then the arguments of
the function which represent the inputs to the element (see Appendix B). The arguments to any of the SIDAC-I elements are not limited to a single variable but may in fact be any expression with the restriction that the arguments which are the control integer and designator must be fixed-point variables or constants and the rest be floating-point variables without subscripts, constants, or expressions. Array names must not be used. The computation is done in floating-point mode. The programmer is urged to check with the listing in Appendix B to make sure that he has all the arguments in the correct modes. The computer fails to operate properly if the arguments are not in the correct format. Equation (4-2) illustrates, as an example, a typical patching format of a simulation instruction.

\[ \text{FG1} = \text{FUNG(XINT, 2, 1)} \]  \hspace{1cm} (4-2)

Equation (4-2) indicates that the function generator, which is referred to by the name FG1, has one independent variable input which comes from block XINT. The fixed-point constant 2 of the argument indicates that the x - y coordinates are stored in function table number 2. The integer 1 of the third input indicates that the function will be generated by the first-order interpolation method (linear interpolation approximation).

It may be noticed that there are three types of arguments that may appear as inputs in any of the SIDAC-I patching instructions.
(1). The argument that stands for continuous variables. It can be a single variable or an arithmetic expression which consists of an algebraic combination of variables names, some numerical quantities, and the four basic operators, i.e., addition, subtraction, division, and multiplication operators. For example,

\[ 3.0 \times \text{DERIV} + \text{FUN1-D2}/(5.0 + \text{DX}). \]  
(4-3)

(2). The argument that stands for designator variables or constants. Examples of designators are numbers referring to function tables and order of interpolation. These numbers are normally fixed-point constants.

(3). The argument that stands for logical variables. This can be either a single variable or an expression. When a component receives a logical input it makes a distinction only between positive and non-positive (including zero) quantities. They must be floating-point variables or constants.

The above argument is valid only for the patching of SIDAC-I elements which are listed in Appendix B. For those FORTRAN functions such as Sine function, Natural logarithm, Absolute Value, etc., the programmer is expected to follow the rules of FORTRAN (4, 9, 12).

**Data Deck.** The data deck consists of the following:

(1). Control Card
This must be the first card of the data package. The three ten column fields using columns 11 to 40 specify the values of the integration interval in field 1 (columns 11-20), the print interval in field 2 (columns 21-30), and the termination time in field 3 (columns 31-40). All three fields are floating-point numbers.

(2). Initial Condition Values

This is for the initial condition values of simple integrators only. The order in which the initial condition values are punched in a card must agree with the order of integrators in the program. If all integrators have zero initial conditions, this item may be omitted.

A designator card is necessary to load the initial conditions into the program. Thus the first card of this item must consist of the word IC in columns 11-12, and the number of simple integrators which are being used in the program in columns 21-22. This designator must be an integer. The initial condition values are punched in a seven ten-column field card using columns 1-70 inclusive. The format is 7F10.4. The number of cards which contain the initial conditions is limited by the maximum number of simple integrators which is 20. Thus the programmer may use as many as three cards to list all initial condition values of the simple integrators in a single problem.

The initial condition values of mode controlled integrators are placed in the patching statements as inputs. Adjustment of the initial
conditions of a mode controlled integrator may be made by employing a PARAMETER statement. In that case, the initial condition values have to be in the PARAMETER data cards.

(3). Parameter Cards

This item is omitted if there is no PARAMETER statement being used in the program. The first card has to be a designator card with the word PA punched in columns 11-12 and the number of parameters (an integer) punched in columns 21-22. The parameter values are punched in a seven ten-column field card using columns 1-70 inclusive. The format is 7F10.4. The number of cards can be any number from one to three.

(4). Function Tables Data Cards

A maximum of three tables may be used. The first card must contain the word TA in columns 11-12 and the table number (an integer) in column 22. The second card must contain only the number of x - y coordinates pairs that are to be used to generate the function. This number must be an integer and appears in columns 1-2. Up to 20 points may be used. Each of the remaining cards can contain up to six values representing the x - y coordinates of up to three points. They appear in floating-point form with each of them occupying a field of ten columns. The abscissa values must be confined within columns 1-10, 21-30, and 41-50, while the ordinates must be confined within columns 11-20, 31-40, and 51-60.
To pick up these $x - y$ coordinates it is necessary to select the first point to be below the lowest anticipated value of the independent variable of the function. Each successive set of coordinates must be listed in order of the increasing algebraic value of the independent variable.

In the case the second and/or the third tables are also used, the data cards will bear the same format as the first table except for the appropriate change of the table number.

5. Print Out Headings Control Card

This item causes the information listed in the Header statement to be typed at the top of the output sheet. The first card consists of the word HD in columns 11-12 and an integer shows the number of cards in columns 21-22. As many heading cards may be used as are needed to list the information. Each heading card is capable of accepting 72 characters starting from column 1 and ending with column 72. The characters can be numeric and/or alphanumeric; blanks are included.

6. End Card

A blank card has to be submitted at the end of the data deck to denote the end of one set of data. The blank card must be present although there might not be any data following the control card.
Procedure of Setting Additional Runs and Reinitialization

The reader is reminded that SIDAC-I is capable of executing three consecutive problems in a single loading. Thus after one program has been completely executed the computer will then come to a halt and wait for the next command from the operator. At this point the operator may:

1. give a command to the computer to start one of the new programs which has been stored in the memory,

2. add more computation time to the same problem in order to obtain more information from that particular problem,

3. change the initial conditions and/or parameter values and rerun the same problem with a new set of data.

The following are the procedures for setting up the computer for the three cases.

Case 1. To start a new program which has been already stored in the memory the operator may do the following:

1. Turn on Sense Switch 2 and press the START key on the computer console. The computer will than automatically type out the following message:

   ENTER NEW PROGRAM

   ( ) PROGRAM NUMBER
(2). Turn the typewriter paper roller back one line and insert the program number of the program to be run within the parenthesis. Press the RS (release and start) key. It may be noted that the program number is an integer 1, 2, or 3 which corresponds to the integer \( n \) in the statement 

\[
\text{SUBROUTINE SIDACn.}
\]

(3). Read into the computer a new complete set of data which contains a time control card and other items as required by the new program. When the plot-version is used the typewriter automatically types another line of message:

\[
\text{TURN ON SENSE SWITCH 3 FOR NEW PLOT FRAME}
\]

The operator then turns on Sense Switch 3 if he wants the new plot frame to be drawn; he leaves it off otherwise.

Case 2. This is the case where some additional running time is required for the same problem with the same set of data, and the programmer wants the computation to carry on from a point starting with the previous termination time. The operator may do the following:

(1). Place a new control card which contains the integration interval, the print interval, and the additional time into the card-reader hopper. The card format is the same as the previous control card.

(2). Turn on Sense Switch 4.
(3). Press the START key on the console to continue the computation.

Case 3. In this case, the same problem is run with a new set of initial conditions and/or new parameter values. The operator may do the following:

(1). Turn on Sense Switch 1. When the plot-version is used the computer automatically types out the following:

TURN ON SENSE SWITCH 3 FOR NEW PLOT FRAME

The operator then turns on Sense Switch 3 if he wants the new plot frame to be drawn; otherwise he leaves it off.

(2). Read into the computer a new complete set of data which contains a control card, and some other items which are required by the program.

Procedure for Interrupting the Run

The programmer might find himself in the position where he was forced to change the integration period, the printed interval, or even some initial conditions and/or parameter values while the program is still being executed. In any event, the computation can be interrupted at any instant and the sense switches can be set in accordance with the operational option one desires.

(1). Momentarily turn on Sense Switch 4 on the console. This will cause the computer to stop.
(2). Turn on Sense Switch 1 if the computation is to be re-
started at the beginning.

(3). Turn on Sense Switch 4 to enter the new time control
information. The computation is to be carried on from
the point where the interruption was made. This case
is applied only when Sense Switch 1 was not previously
turned on.

(4). Press the START key on the computer console to con-
tinue the computation. If steps 2 and 3 are not executed
and step 4 is executed after Sense Switch 4 is turned off,
the effect is then the same as pressing the INSTANT
STOP key on the computer console for momentarily stop.

**Deck Arrangement and Compilation Procedure**

To implement the SIDAC-I program on the IBM 1620 computer
it is assumed that the user has a prior knowledge of 1620 operating
principles. The user is urged to consult with the 1620 operation
manual (4, 7) for the information necessary for producing a
FORTRAN OBJECT program. A complete SIDAC-I program deck
consists of:

(1). SIDAC-I main control deck.

(2). SUBROUTINE SIDACn deck which specifies the con-
figuration of a simulation system.
(3). SIDAC-I predefined functions deck which contains all the functions necessary for computing the problem.

(4). FORTRAN subroutines deck.

(5). Data Deck.

All but the SUBROUTINE SIDACn deck are available in the form of object programs. Before actual calculation the SUBROUTINE SIDACn deck must be translated into the machine language program (object program). There are two different compilers that can be used for this translation: the plot compiler and the regular FORTRAN II batch compiler. When the plot-version of SIDAC-I is used the translation has to be done by the plot-compiler. In the case that a IBM 1627 plotter is not present, the no-plot version of SIDAC-I must be used and the compilation of SUBROUTINE SIDACn program is done by the regular FORTRAN II batch compiler.

The object programs are executed in the order shown in Figure 18.

Figure 18. A complete deck of the SIDAC-I program.
It was mentioned previously that a multiple run can be arranged by loading \( n \) \textsc{SUBROUTINE SIDACn} object programs consecutively where \( n \) is any positive integer not greater than three. In other words, a maximum of three different problems can be read into the computer in a single loading. This is obviously a time saving process when more than one problem is to be run. In order to have this property the SIDAC-I main control was designed to receive a fixed number of three \textsc{SUBROUTINE SIDACn} programs in every loading. Thus one or more dummy \textsc{SUBROUTINE SIDACn} decks are required to replace the missing \textsc{SUBROUTINE SIDACn} program(s), i.e., if there were only two programs \textsc{SUBROUTINE SIDAC1} and \textsc{SUBROUTINE SIDAC2} to be run then the dummy \textsc{SUBROUTINE SIDAC3} has to be read into the computer together with the other two programs. The three dummy \textsc{SUBROUTINE SIDACn} programs are available in the form of machine language programs.

Sometimes the programmer will find it difficult to keep track of the predefined functions which were used in a specific problem. As far as loading the subroutines of the predefined functions of a large problem into the computer is concerned the programmer will find it quite confusing. It is then strongly recommended, although it is not necessary, to divide the loading of a complete deck of a SIDAC-I program into two parts. The first part should consist of the SIDAC-I main control deck and the \textsc{SUBROUTINE SIDACn}
programs. The rest belongs to the second part. As a result of completing the loading of the first portion the computer automatically types a message:

A list of function names

READ SUBPROGRAMS NAMED ABOVE

The names shown in the list are the names of the predefined functions which are required in the SUBROUTINE SIDACn programs. The operator then sorts out the object programs of these functions and loads them into the computer together with the rest of the deck.

Sense Switches

There are four sense switches on an IBM 1620 computer console. They are used by the SIDAC-I simulator to achieve the same measure of control provided by the usual "Standby-Initial Condition-Operate" switch of the analog computer. The important roles of these sense switches have been previously discussed in various parts of this paper. A summary of the sense switch settings for the various operations is as follows.

(1). Sense Switch 1 on permits the operator to modify both
the time control and initial/parameter specifications.

(2). Sense Switch 2 on permits the operator to enter the new program number of the program which has been previously stored in the memory.

(3). Sense Switch 3 on causes the plotter to move the paper beyond the present plot and prepare a new plot frame. This switch is used only in the plot-version of SIDAC-I.

(4). Sense Switch 4 on permits the operator to enter the additional operating time for the present problem.

The above four cases cannot be applied unless Sense Switch 4 was momentarily turned on or the computer has stopped due to the termination time. After the computation was terminated due to either of the two cases, the operator then sets the sense switches in accordance with the operational option he desires and presses the START key on the console to continue.

**Error Messages**

Any invalid patching instruction may cause the computer to type out an error message. This is the error which can be detected by
the SIDAC-I elements. The error messages have the following form:

\[ \text{SIDAC ERROR } n \]

where \( n \) is an integer which identifies the type of error. At the present time the SIDAC-I simulator is capable of detecting only 15 different types of errors. They are listed as follows:

1. **SIDAC ERROR 1**  
The designator integer in the function table data card which shows the function table number is out of range (it is invalid when it is greater than 3) or the program number is out of range (invalid when it is greater than 3).

2. **SIDAC ERROR 2**  
Error is in a Parameter instruction statement. It could be that the number of parameters in a single Parameter statement is out of the allowable range or the total number of parameters exceeds its allowable range (not valid when it is greater than 20).

3. **SIDAC ERROR 3**  
Error is in the Type-Output statement. It could be that the number of variable values to be typed exceeds the allowable range (not valid when it is greater than 5).

4. **SIDAC ERROR 4**  
Error is in the Punch-Output statement. It could be that the number of variable values to be punched exceeds the allowable range.
5. SIDAC ERROR 5  Error is in the Flip-flop instruction statement. It could be that the number of flipflops which are used in a single problem exceeds the allowable range (not valid when the number is greater than 5).

6. SIDAC ERROR 6  Error is in the Track-Transfer instruction statement. It could be that the number of the Track-Transfers which are used in a problem exceeds its allowable range (not valid when the number is greater than 10).

7. SIDAC ERROR 7  Error is in the Function Generator patching statement. It could be that the number of the function table appears in the patching statement does not correspond with the function table which stores the x-y coordinates.

8. SIDAC ERROR 8  Error is in the Pulse Generator patching statement. It could be that the number of pulse generators which are used in the program exceeds the allowable range (not valid when the number is greater than 5).

9. SIDAC ERROR 9  Error is in the Sawtooth Generator patching statement. It could be that the number of
sawtooth generators which are used in the program exceeds the allowable range (not valid when the number is greater than 5).

10. SIDAC ERROR 10 Error is in the Square Wave Generator patching statement. It could be that the number of square-wave generators which are used in the program exceeds the allowable range (not valid when the number is greater than 5).

11. SIDAC ERROR 11 Error is in the Mode Controlled Integrator patching statement. It could be that the number of mode controlled integrators which are used in the program exceeds its allowable range (not valid when the number is greater than 20).

12. SIDAC ERROR 12 Error is in the Simple Integrator patching statement. It could be that the number of simple integrators which are used in the program exceeds its allowable range (not valid when it is greater than 15).

13. SIDAC ERROR 13 Error is in the Counter patching statement. It could be that the number of counters which are used in the program exceeds its allowable range (not valid when it is greater than 10).
14. SIDAC ERROR 14 Error is in the Delay patching statement. It could be that the number of delays which are used in the program exceeds its allowable range (not valid when the number is greater than 20).

15. SIDAC ERROR 15 Error is in the Zero-Order Hold patching statement. It could be that the number of zero-order hold elements which are used in the program exceeds its allowable range (not valid when the number is greater than 10).

Whenever one of the SIDAC error messages is typed out the execution of the program is terminated. In most cases the program has to be corrected and then recompiled. However, if the error was caused by the invalid program number the operator will have only to reset the computer and insert the correct program number.
EXAMPLES

Example 1: SIDAC-I Solution of the Van der Pol Equation

The Van der Pol nonlinear differential equation has the form

\[ \frac{d^2 X}{dt^2} + C_1 f(X) \frac{dX}{dt} + w_0^2 X = 0 \]  

(5-1)

where \( f(X) \) is the nonlinear term. For almost sinusoidal oscillations, the nonlinear term must remain small over the range of interest.

Let us consider the case where the nonlinear term is given by the equation

\[ f(X) = X^2 - 1 \]  

(5-2)

and the frequency of oscillation \( w_0 = 1 \). Thus the oscillator equation becomes

\[ \frac{d^2 X}{dt^2} + C_1 (X^2 - 1) \frac{dX}{dt} + X = 0 \]  

(5-3)

This example demonstrates how the PARAMETER statement may be used to change the parameter of the system. \( C_1 \) is the parameter which has the value 0.1 to start with; it is changed to 1.0 and 3.0 respectively.

By following the step-by-step procedure given in the previous section of this paper the result is given as follows:
(1). The equation with the highest derivative on the left

\[ \dddot{X} = -C_1(X^2 - 1)X - \dot{X} \]  

(5-4)

The initial conditions are \( x(0) = 1; \dot{x}(0) = 0.0001 \).

(2). The block diagram for the equation is shown in Figure 19.

![Block diagram for the Van der Pol equation](image)

Figure 19. Block diagram for the Van der Pol equation.

(3). The SIDAC-1 program is written as follows:

```c
C SIDAC-1 SOLUTION OF THE VAN DER POL EQUATION
SUBROUTINE SIDAC1(I, X2, XM2, D1, D2)
CALL PAR(1, C1, D, D, D, D)
T=CLOCK(1.0)
X1=SINT(-X2-XM2*C1)
X2=SINT(X1)
XM1=X2*X2
```

SUN=XM1-1.0
XM2=SUM*X1
CALL TYOUT(3, T, X1, X2, 0.0, 0.0)
CALL PHOUT(3, 1, T, X1, X2, 0.0, 0.0)
RETURN
END

*DATA

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2.0</td>
<td>16.0</td>
</tr>
<tr>
<td>IC</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0.0001</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>PA</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>2.0</td>
<td>16.0</td>
</tr>
<tr>
<td>IC</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0.0001</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>PA</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>2.0</td>
<td>16.0</td>
</tr>
<tr>
<td>IC</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0.0001</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>PA</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note that the data deck consists of three sets of data. Each has the different parameter value of 0.1, 1.0, and 3.0 respectively. A blank card is required for each set as the end card.

To enter the new parameter after the first run the operator turns on Sense Switch 1 and presses the START key. The second set of data is then read into the computer and the computation continues with the new parameter value. The typewriter output data for the three cases is recorded in the following page and is plotted in Figure 21.

Example 2. Hybrid Solution of an Optimization Control Problem

Let us consider a system whose transfer function is

\[ G(s) = \frac{1}{S^2 + 2zS + 1} \]  

(5-5)

A unit step input is applied to this system at \( t = 0 \). We are interested in finding the value of the damping factor \( z \) for an overshoot of 10 percent. Assume an error of plus or minus five percent.

The logical operation is required to search for the value of \( z \) which gives the desired output. Initially, \( z \) is assigned to have the value 1.0. If the response has no overshoot or if the overshoot is less than 10.0 plus or minus five percent the damping factor \( z \) is decreased by a factor of \( DZ/N \) where \( N \) was the number of trials.

On the contrary, if the overshoot is greater than 10.0 plus or minus
SIDAC-I A SIMULATED DIGITAL-ANALOG (HYBRID) COMPUTER

1PS PROGRAM NUMBER

TIME INCREMENT = .10000

PARAMETER C1=0.1

<table>
<thead>
<tr>
<th>TIME</th>
<th>X1</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>.00010</td>
<td>.10000</td>
</tr>
<tr>
<td>2.0000</td>
<td>-.10068</td>
<td>-.05099</td>
</tr>
<tr>
<td>4.0000</td>
<td>.09265</td>
<td>-.07523</td>
</tr>
<tr>
<td>6.0000</td>
<td>.03761</td>
<td>.13168</td>
</tr>
<tr>
<td>8.0000</td>
<td>-.14795</td>
<td>-.02940</td>
</tr>
<tr>
<td>10.0000</td>
<td>.09034</td>
<td>-.13377</td>
</tr>
<tr>
<td>12.0000</td>
<td>.09743</td>
<td>.15921</td>
</tr>
<tr>
<td>14.0000</td>
<td>-.20009</td>
<td>.01672</td>
</tr>
<tr>
<td>16.0000</td>
<td>.06550</td>
<td>-.20968</td>
</tr>
</tbody>
</table>

PARAMETER C1=1.0

<table>
<thead>
<tr>
<th>TIME</th>
<th>X1</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>.00010</td>
<td>.10000</td>
</tr>
<tr>
<td>2.0000</td>
<td>-.30716</td>
<td>-.19761</td>
</tr>
<tr>
<td>4.0000</td>
<td>.33712</td>
<td>-.46629</td>
</tr>
<tr>
<td>6.0000</td>
<td>.46840</td>
<td>1.42234</td>
</tr>
<tr>
<td>8.0000</td>
<td>-2.19196</td>
<td>-.18158</td>
</tr>
<tr>
<td>10.0000</td>
<td>.74723</td>
<td>-.53312</td>
</tr>
<tr>
<td>12.0000</td>
<td>1.18858</td>
<td>1.82376</td>
</tr>
<tr>
<td>14.0000</td>
<td>-1.26414</td>
<td>.88135</td>
</tr>
<tr>
<td>16.0000</td>
<td>.40727</td>
<td>-.92784</td>
</tr>
</tbody>
</table>

PARAMETER C1=3.0

<table>
<thead>
<tr>
<th>TIME</th>
<th>X1</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>.00010</td>
<td>.10000</td>
</tr>
<tr>
<td>2.0000</td>
<td>-.85634</td>
<td>-.48692</td>
</tr>
<tr>
<td>4.0000</td>
<td>1.13064</td>
<td>-.76885</td>
</tr>
<tr>
<td>6.0000</td>
<td>-.24895</td>
<td>1.83780</td>
</tr>
<tr>
<td>8.0000</td>
<td>-.64005</td>
<td>1.10461</td>
</tr>
<tr>
<td>10.0000</td>
<td>.21870</td>
<td>-.93286</td>
</tr>
<tr>
<td>12.0000</td>
<td>.45648</td>
<td>-.31976</td>
</tr>
</tbody>
</table>

Figure 20. The typewriter output for the Van der Pol equation problem.
Figure 21. Graphical output for the Van der Pol equation problem.
five percent the damping factor is increased by a factor of $\frac{DZ}{N}$.

The SIDAC-I program consisted of two portions. The first portion is the analog program representing the dynamic equation of the system;

$$S^2 Y = 1.0 - 2zSY - Y \quad (5-6)$$

Note that the PARAMETER statement has to be used because there are variables in the program; they are the damping factor $z$, the number of trials $N$, the control signal $A$ which has to be generated to reset the mode controlled integrators, the increment damping factor $DZ$, and finally the control variable $P$ which controls the execution of the analog program and the logical program.

The following pages show the complete listing of the program, the block diagram, and the typewriter record of the output.

Example 3. Hybrid Solution of a Optimization Control Problem

Repeat Example 2 using the SIDAC-I block-oriented language to replace the FORTRAN language for the logical portion of the program. Initially, $z$ is assigned to have the value of 0.7.

The block diagram is shown in page 89. The analog program has the same form as in Example 2. The elements which are used to generate the REP-OP control signals for the mode controlled integrators and the gating control signals are: the mode controlled
SUBROUTINE SIDAC1(I,SY,Y,D1,D2)

ANALOG PROGRAM
CALL PAR(5,Z,A,P,DZ,AN)
TIME=CLOCK(1.0)
SY=CINT(0.0,A+1.0,0.0-2.0*Z*SY-Y)
Y=CINT(0.0,A+1.0,SY)
CONT=CINT(0.0,A+1.0,1.0)
CALL QUIT(CONT,6.0)
CALL TYOUT(4,TIME,CONT,SY,Y,DUMMY)
XAX=OFFSET(5.0*TIME,-100.0)
YAX=OFFSET(200.0/3.0*Y,-100.0)
CALL PLOTER(XAX,YAX)

LOGICAL PROGRAM
IF (I) 1,1,10
1 IF (SY+0.0001) 4,4,2
2 IF (CONT-5.0) 3,4,4
3 A=1.0
RETURN
4 IF (P-1.0) 5,3,5
5 IF (Y-1.101) 6,8,8
6 IF (Y-1.099) 9,9,7
7 P=1.0
TYPE 11,Z
RETURN
8 AN=AN+1.0
A=0.0
Z=Z+DZ/AN
TYPE 12,Z
RETURN
9 AN=AN+1.0
A=0.0
Z=Z-DZ/AN
TYPE 12,Z
10 RETURN
11 FORMAT(29HTHE DESIRED DAMPING FACTOR Z=F10.5)
12 FORMAT(10HTHE NEW Z=F10.5)
END

* DATA
  0.1  2.0  80.0
  PA  5
  1.0  1.0  0.0  0.3  0.0
Figure 22. Simulation diagram for optimization feedback control problem.
SIDAC -I A SIMULATED DIGITAL-ANALOG (HYBRID) COMPUTER

(1RS PROGRAM NUMBER)

TIME INCREMENT = .10000

<table>
<thead>
<tr>
<th>TIME</th>
<th>CONTROL</th>
<th>DY</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>2.00000</td>
<td>2.00000</td>
<td>1.27014</td>
<td>0.59403</td>
</tr>
<tr>
<td>4.00000</td>
<td>4.00000</td>
<td>0.07338</td>
<td>0.90816</td>
</tr>
<tr>
<td>THE NEW Z=</td>
<td>.70000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.00000</td>
<td></td>
<td>1.44926</td>
<td>0.26087</td>
</tr>
<tr>
<td>8.00000</td>
<td></td>
<td>1.16064</td>
<td>0.95055</td>
</tr>
<tr>
<td>THE NEW Z=</td>
<td>.55000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.0000</td>
<td></td>
<td>1.26356</td>
<td>0.03727</td>
</tr>
<tr>
<td>12.0000</td>
<td></td>
<td>1.35121</td>
<td>1.00844</td>
</tr>
<tr>
<td>THE NEW Z=</td>
<td>.45000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.0000</td>
<td></td>
<td>1.40989</td>
<td>0.13881</td>
</tr>
<tr>
<td>16.0000</td>
<td></td>
<td>1.23030</td>
<td>0.97766</td>
</tr>
<tr>
<td>THE NEW Z=</td>
<td>.58500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.0000</td>
<td></td>
<td>1.39928</td>
<td>0.13649</td>
</tr>
<tr>
<td>20.0000</td>
<td></td>
<td>1.22399</td>
<td>0.93942</td>
</tr>
<tr>
<td>THE NEW Z=</td>
<td>.59214</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22.0000</td>
<td></td>
<td>1.31500</td>
<td>0.06520</td>
</tr>
<tr>
<td>24.0000</td>
<td></td>
<td>1.27909</td>
<td>0.92119</td>
</tr>
<tr>
<td>THE DESIRED DAMPING FACTOR Z=</td>
<td>.59214</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26.0000</td>
<td></td>
<td>1.03608</td>
<td>1.08884</td>
</tr>
</tbody>
</table>

Figure 23. The typewriter output for Example 2.
Figure 24. The graphical output for Example 2.
integrator, AINT3; the limiters, ALM1, ALM2, and ALM3; and the NOT gates, ANT1 and ANT2. The elements which are used to test the overshoot of the response are: the limiters, ALM4, ALM5, and ALM6; the flipflops, AFF1, AFF2, and AFF3; the NOT gates, ANT3 and ANT4; and the AND gates, AN1 and AN2. The elements which are used to terminate the computation after the desired response has been obtained are: the NOT gate, ANT5; the NAND gate, ANNI; and the automatic stop. The counter, ACT1, is used to count the number of trials. The divider, DIV1, and the mode controlled integrator, Z, together with the counter are used to adjust the damping factor z.

The complete listing of the S1DAC-I coding of this Example and the typewriter record of the output are shown in the following pages. The output is plotted in Figure 27.
SIDAC CODING OPTIMIZATION EXAMPLE
HYBRID SIMULATION: ADJUSTING THE DAMPING FOR 10 PER CENT OVERSHOOT OF A SECOND-ORDER SYSTEM

SUBROUTINE SIDAC1(I,ANT1,AINT1,AINT2,Z)
TIME=CLOCK(1.0)
AINT3=CINT(0.0,ANT1,1.0,0.0)
ALM1=SLIMIT(AINT3-6.0,1.0,0.0)
ANT1=NOT(ALM1)
AINT2=SLIMIT(AINT3-5.0,1.0,0.0)
ANT2=NOT(ALM2)
ALM3=SLIMIT(AINT3-4.0,1.0,0.0)
AINT1=CINT(0.0,ANT2,1.0,0.0-2.0*Z*AINT1-AINT2)
AINT4=CINT(0.0001,ANT2,1.0,AINT1)
AOR1=OR(ALM4,ALM3,0.0,0.0,0.0)
AFF1=FF(AOR1,ALM1,0.0)
ANT3=NOT(AFF1)
ALM5=SLIMIT(AINT2-1.1001,1.0,0.0)
AFF2=FF(ALM5,ALM1,0.0)
ANT4=NOT(AFF2)
AN1=AND(ANT4,ANT3,1.0,0.0,1.0)
ALM6=SLIMIT(AINT2-1.0999,1.0,0.0)
AFF3=FF(ALM6,ALM1,0.0)
AN2=AND(AFF3,ANT3,1.0,0.0,1.0)
CT1=COUNT(AINT2-0.0003,15.0)
IF (CT1) 1*1*2
2 DIV1=0.3*(AN1-AN2)/CT1
GO TO 3
1 DIV1=0.3
3 CONTINUE
Z=CINT(0.7,ALM2,0.0,0.0,DIV1)
ANT5=NOT(AFF3)
ANN1=NAND(ANT5,AFF2,ANT3,1.0,0.0)
CALL QUIT(-2.0*ANN1+1.0,0.0)
CALL TYOUT(5.0,TIME,AINT3,Z,AINT1,AINT2)
XAXIS=OFFSET(5.0*TIME,-100.0)
YAXIS=OFFSET(200.0/3.0*AINT2,-100.0)
CALL PLOTER(XAXIS, YAXIS)
RETURN
END
* DATA
0.1 4.0 80.0
Figure 25. Block diagram for Example 3.
SIDAC I-A SIMULATED DIGITAL-ANALOG (HYBRID) COMPUTER

(1RS PROGRAM NUMBER

TIME INCREMENT = .10000

<table>
<thead>
<tr>
<th>Time</th>
<th>AINT3</th>
<th>Z</th>
<th>AINT1</th>
<th>AINT2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>0.0000</td>
<td>.70000</td>
<td>0.00000</td>
<td>.00010</td>
</tr>
<tr>
<td>4.0000</td>
<td>4.0000</td>
<td>.70000</td>
<td>.02418</td>
<td>1.04103</td>
</tr>
<tr>
<td>8.0000</td>
<td>1.80000</td>
<td>1.00000</td>
<td>.28372</td>
<td>.56704</td>
</tr>
<tr>
<td>12.0000</td>
<td>5.80000</td>
<td>.88000</td>
<td>0.00000</td>
<td>.00010</td>
</tr>
<tr>
<td>16.0000</td>
<td>3.60000</td>
<td>.85000</td>
<td>.07578</td>
<td>.95138</td>
</tr>
<tr>
<td>20.0000</td>
<td>1.40000</td>
<td>.75000</td>
<td>.41097</td>
<td>.51546</td>
</tr>
<tr>
<td>24.0000</td>
<td>5.40000</td>
<td>.72000</td>
<td>0.00000</td>
<td>.00010</td>
</tr>
<tr>
<td>28.0000</td>
<td>3.20000</td>
<td>.67500</td>
<td>.09427</td>
<td>1.01815</td>
</tr>
<tr>
<td>32.0000</td>
<td>1.00000</td>
<td>.61500</td>
<td>.49342</td>
<td>.36945</td>
</tr>
<tr>
<td>36.0000</td>
<td>5.00000</td>
<td>.61500</td>
<td>-.04223</td>
<td>1.05340</td>
</tr>
<tr>
<td>40.0000</td>
<td>2.80000</td>
<td>.56500</td>
<td>.15918</td>
<td>1.05248</td>
</tr>
<tr>
<td>44.0000</td>
<td>.60000</td>
<td>.60785</td>
<td>.43766</td>
<td>.18017</td>
</tr>
<tr>
<td>48.0000</td>
<td>4.60000</td>
<td>.60785</td>
<td>-.04017</td>
<td>1.07177</td>
</tr>
<tr>
<td>52.0000</td>
<td>2.40000</td>
<td>.57035</td>
<td>.25803</td>
<td>.96497</td>
</tr>
<tr>
<td>56.0000</td>
<td>.20000</td>
<td>.60369</td>
<td>.25313</td>
<td>.03653</td>
</tr>
<tr>
<td>60.0000</td>
<td>4.20000</td>
<td>.60369</td>
<td>-.02652</td>
<td>1.08711</td>
</tr>
<tr>
<td>64.0000</td>
<td>2.00000</td>
<td>.57369</td>
<td>.36116</td>
<td>.83852</td>
</tr>
<tr>
<td>68.0000</td>
<td>6.00000</td>
<td>.60096</td>
<td>0.00000</td>
<td>.00010</td>
</tr>
<tr>
<td>72.0000</td>
<td>3.80000</td>
<td>.60096</td>
<td>.00256</td>
<td>1.09395</td>
</tr>
<tr>
<td>76.0000</td>
<td>1.60000</td>
<td>.57596</td>
<td>.45197</td>
<td>.67363</td>
</tr>
<tr>
<td>80.0000</td>
<td>5.60000</td>
<td>.58980</td>
<td>0.00000</td>
<td>.00010</td>
</tr>
</tbody>
</table>

Figure 26. The typewriter output for Example 3.
Figure 27. The graphical output for Example 3.
CONCLUSIONS

The objective of developing the SIDAC-I simulation program has not been restricted to providing for only the availability of software to solve differential equations for numerical solutions on a comparatively small, scientific, digital computer. The main objective has been to demonstrate a way of including a computer language as an integral part of the SIDAC-I simulation language. The end result is that SIDAC-I permits one or more FORTRAN II statements in the simulation program. This feature of SIDAC-I permits extensive logical and functional operations to be performed on the simulation data during the runs. The numerical solution of the analog simulation can be tested by the digital computer for parameter optimization, response criterion, or any other form of preselected mode of system testing that was specified by the programmer. In brief, SIDAC-I simulator permits the hybrid as well as analog simulations to be programmed on a digital computer by using the SIDAC-I block-oriented language combined with the FORTRAN II language.

An efficient simulator program should be able to provide a "feel" to the programmer for his simulation. It should include some kind of capabilities to allow the programmer to "play" with his system and to create a learning process from a cause-effect relationship in system response (1). It is true that SIDAC-I is able to
perform logical operation which makes itself a good tool for some small synthesis problems; but due to the fact that it was developed for a small computer with the limited amount of storage, many of these "nice" features which can be incorporated with a larger computer must be eliminated. The end result of this "must" caused SIDAC-I, inevitably, a few drawbacks: First, SIDAC-I does not provide sufficient "hand-on" control of operation of the program; secondly, it lacks of the ability of adding new functions external to the basic program; Thirdly, it lacks of the ability to diagnose the configuration of the simulation program. Its ability of detecting errors from the system is poor. Further study should be made along this line to improve these shortcomings. The program redundancy is undoubtedly great within the substructures of the simulator. This redundancy should be eliminated as much as possible for the sake of saving computation time as well as the core storage.
BIBLIOGRAPHY


APPENDICES
APPENDIX A

THE DEFINITION OF TURING MACHINE

The Turing machine is an automatic calculating device which
was invented by A. M. Turing in 1936 (16, p. 231). This device is
essentially a black box which is capable of assuming a specified
finite number of internal states. It operates on a tape with infinite
length by writing in letters or symbols. Each of these symbols or
letters occupies a square on this tape. The machine is capable of
viewing only one square at a time. Its behavior is completely de-
termined by the symbol on the scanned square and the internal state.
The machine changes the content of the scanned square, or it is
normally called as a scanned cell, to some other specified symbol
or alphabet letter. It is capable of making a move to the left or the
right of the scanned cell, or it may confine its attention to the cell
it has examined.

A detailed definition of a Turing machine can be found in the
booklet written by B. A. Trakhtenbrot (15, p. 58-61).
## APPENDIX B

### Description of SIDAC-I Elements

<table>
<thead>
<tr>
<th>Name-Symbol</th>
<th>Patching Instruction</th>
<th>Output (typical)</th>
<th>Core Storage requirements FORTRAN II</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Automatic Stop</strong></td>
<td>CALL QUIT(X1, X2)</td>
<td>Machine stops when X1 &gt; X2.</td>
<td>310</td>
<td>Accepts 2 inputs. Both X1 and X2 are floating point variables or constants.</td>
</tr>
</tbody>
</table>
| **AND Gate** | $Y = \text{AND}(X1, X2, X3, X4, X5)$ | $Y = 1.0$ if $X_i \neq 0$ for $i=1, 2, 3, 4, 5$; $Y = 0.0$ otherwise. | 810 | 1. Accepts up to 5 inputs.  
2. All inputs are floating-point variables or constants.  
3. Dummy variable(s) are required to replace the unused input(s). Dummy variables are assigned a value of 1.0. |
| **Bang-Bang** | $Y = \text{BANG}(X, P)$ | $Y = +P$ if $X \geq 0$  
$= -P$ if $X < 0$ | 440 | Accepts only one input. X and P are floating-point variables or constants. This is equivalent to $Y = \text{PsgnX}$. |
<table>
<thead>
<tr>
<th>Name-Symbol</th>
<th>Patching Instruction</th>
<th>Output (typical)</th>
<th>Core Storage requirements FORTRAN II</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOCK</td>
<td>( T = \text{CLOCK}(A1) )</td>
<td>The output is the current value of some integral multiple of the current value of the independent variable.</td>
<td>214</td>
<td>One floating-point constant argument. This function is always present in the program.</td>
</tr>
<tr>
<td>Comparater</td>
<td>( Y = \text{COMP}(X_1, X_2) )</td>
<td>( Y = 1.0 ) if ( X_1 &gt; X_2 ) ( Y = 0.0 ) if ( X_1 \leq X_2 )</td>
<td>402</td>
<td>There are two arguments. ( X_1 ) and ( X_2 ) are floating-point variables or constants. Comparison between two analog inputs yields one digital output.</td>
</tr>
<tr>
<td>Counter</td>
<td>( Y = \text{COUNT}(X, P) )</td>
<td>( Y = ) number of times the input ( X ) is greater than zero.</td>
<td>2024</td>
<td>A maximum of 10 counters may be used. These are zero crossing counters. ( X ) and ( P ) are floating-point variables or constants. Modulo the integer part of input ( P ), i.e., if ( P = 5.6 ), the modulo = 5.</td>
</tr>
<tr>
<td>Name-Symbol</td>
<td>Patching Instruction</td>
<td>Output (typical)</td>
<td>Core Storage requirements FORTRAN II</td>
<td>Remarks</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
<td>-----------------</td>
<td>--------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Dead Space</td>
<td>Y = DEADS(X, UB, LB)</td>
<td>Y = 0 when LB ≤ X ≤ UB.&lt;br&gt;Y = X - LB when X &lt; LB.&lt;br&gt;Y = X - UB when X &gt; UB.</td>
<td>640</td>
<td>The threshold gap has a length of UB+LB. X, UB, and LB are floating-point variables or constants. UB and LB are the break points of a threshold nonlinearity. UB &gt; LB.</td>
</tr>
<tr>
<td>Delay</td>
<td>Y = DELAY(X)</td>
<td>Y = X(t - Δt/2) or Y = X exp(-sΔt/2)</td>
<td>1282</td>
<td>Each delays the arrival of input X by one half of an integration period. Delay of longer time can be achieved by connecting a number of delay elements in series. A maximum of 20 delays may be used which gives the maximum delay time of 10 integration periods. X is a floating-point variable.</td>
</tr>
<tr>
<td>Name-Symbol</td>
<td>Patching Instruction</td>
<td>Output (typical)</td>
<td>Core Storage requirements FORTRAN II</td>
<td>Remarks</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------</td>
<td>------------------------------------------------------</td>
<td>--------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Flip-Flop</td>
<td></td>
<td></td>
<td></td>
<td>The three logical inputs are floating-point variables or constants. A maximum of 15 flip-flops may be used in a single program. Positive inputs are treated as 1. Negative inputs are treated as 0. This is RST flipflop.</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Y = FF(R, S, T)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complements the output when T &gt; 0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y = 1      if S &gt; 0.</td>
<td>1740</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y = 0      if R &gt; 0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function Generator</td>
<td></td>
<td></td>
<td></td>
<td>M and N are control integers, they must be fixed-point constants. M denotes the number of function table, 1 ≤ M ≤ 3. N denotes the order of interpolation, 1 ≤ N ≤ 2. X is independent variable and must be a floating-point variable. A maximum of 3 function generators may be used. Up to 20 sets of x-y coordinates can be used. Spacing of break points is arbitrary.</td>
</tr>
<tr>
<td>X</td>
<td>Y = FUNG(X, M, N)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y = f(X)</td>
<td>4460</td>
<td></td>
</tr>
<tr>
<td>Name-Symbol</td>
<td>Patching Instruction</td>
<td>Output (typical)</td>
<td>Core Storage requirements FORTRAN II</td>
<td>Remarks</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------</td>
<td>------------------</td>
<td>-------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Integrator (Simple)</td>
<td>Y = SINT(X)</td>
<td>Y = ( \int_{0}^{t} X dt + IC )</td>
<td>1770</td>
<td>The initial condition may be set by an appropriate entry in the IC cards. X is integrand and it must be floating-point variable or constant. A maximum of 15 simple integrators may be used in one program.</td>
</tr>
<tr>
<td><img src="image1" alt="Diagram of Integrator (Simple)" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Integrator (Mode Controlled) | Y = CINT(IC, P1, P2, X) | Y = \( \int_{0}^{t} X dt + IC \) when P1 > 0
Y = IC when P1 \leq 0
P2 > 0
Y = Previous Y
(Hold mode) when P1 \leq 0
P2 \leq 0 | 3560                                | The initial condition may be set by the value of IC input in the patching instruction or by an appropriate entry in the Parameter statement and Parameter data cards. The IC, the controllers P1 and P2, and the integrand X are floating-point variables or constants. A maximum of 20 mode controlled integrators may be used in a single problem. |
<p>| <img src="image2" alt="Diagram of Integrator (Mode Controlled)" /> |                      |                  |                                     |                                                                                                                                         |</p>
<table>
<thead>
<tr>
<th>Name-Symbol</th>
<th>Patching Instruction</th>
<th>Output (typical)</th>
<th>Core Storage requirements</th>
<th>Remarks</th>
</tr>
</thead>
</table>
| Limiter           | \( Y = \text{SLIMIT}(X, P1, P2) \) | \begin{align*}  
  Y &= P1 \text{ if } X > P1. \\  
  Y &= P2 \text{ if } X < P2. \\  
  Y &= X \text{ if } P2 \leq X \leq P1.  
\end{align*} | 622                       | X, P1, and P2 are the inputs. They must be floating-point variables or constants. P1 and P2 are the saturation points and they may be variables. P1 > P2. |
| NAND Gate         | \( Y = \text{NAND}(X_1, X_2, X_3, X_4, X_5) \) | \begin{align*}  
  Y &= 0.0 \text{ if } X_i > 0 \\  
  i &= 1, 2, 3, 4, 5 \\  
  Y &= 1 \text{ otherwise.}  
\end{align*} | 794                       | 1. Accepts up to 5 inputs.  
2. All inputs are floating-point variables or constants.  
3. Dummy variables are required to replace the unused inputs. Dummy variables are assigned as 1.0. |
| Negative Clipper  | \( Y = \text{CLIPN}(X) \) | \begin{align*}  
  Y &= 0 \text{ for } X \leq 0. \\  
  Y &= X \text{ for } X > 0.  
\end{align*} | 316                       | It clips off the negative portion of the input waveform. The input X must be a floating-point variable. |
<table>
<thead>
<tr>
<th>Name-Symbol</th>
<th>Patching Instruction</th>
<th>Output (typical)</th>
<th>Core Storage requirements</th>
<th>Remarks</th>
</tr>
</thead>
</table>
| NOR Gate    | $Y = \text{NOR}(X_1, X_2, X_3, X_4, X_5)$ | $Y = 0$ if any of the inputs is positive. $Y = 1.0$ otherwise. | 794 | 1. Accepts up to 5 inputs.  
2. All inputs are floating-point variables or constants.  
3. Dummy variables are required to replace the unused inputs. Dummy variables are assigned as 0.0. |
| NOT Gate    | $Y = \text{NOT}(X)$ | $Y = 0$ if $X$ is positive. $Y = 1$ otherwise. | 310 | 1. Accepts only one input.  
2. The input $X$ is floating-point variable or constant. |
<p>| Offset      | $Y = \text{OFFSET}(X, P)$ | $X = X + P$ | 278 | Both inputs must be floating-point variables or constants. This element is very helpful in scaling the x-y coordinates when the plotter is used. |</p>
<table>
<thead>
<tr>
<th>Name-Symbol</th>
<th>Patching Instruction</th>
<th>Output (typical)</th>
<th>Core Storage requirements</th>
<th>Remarks</th>
</tr>
</thead>
</table>
| OR Gate     | Y = OR(X1, X2, X3, X4, X5) | Y = 1. if any of the inputs is positive. Y = 0. otherwise. | 810          | 1. Accepts up to 5 inputs.  
2. All inputs are floating-point variables or constants.  
3. Dummy variables are required to replace the unused inputs. Dummy variables are assigned as 0. |
| X1          |                      |                  |                           |         |
| X4          |                      |                  |                           |         |
| X5          |                      |                  |                           |         |
| Parameter   | CALL PAR(N, P1, P2, P3, P4, P5) | No output        | 1990                      | 1. N, a designator, indicates the number of parameters appear in a single calling statement, 1 ≤ N ≤ 5  
2. Dummy variables are required to replace the unused inputs.  
3. N is a fixed-point constant. All the other inputs are floating-point variable names.  
4. The variable names must be the same in the program if they are meant to represent the same thing.  
5. A maximum of 20 parameters is allowed in a program. |
<table>
<thead>
<tr>
<th>Name-Symbol</th>
<th>Patching Instruction</th>
<th>Output (typical)</th>
<th>Core Storage requirements FORTRAN II</th>
<th>Remarks</th>
</tr>
</thead>
</table>
| PLOTER            | CALL PLOTER(X, Y)    | Output data obtained in graphical form.               | 1100                                 | 1. x-y coordinates have to be scaled properly (see scaling procedure in pages 40-1).  
<p>|                   |                      |                                                       |                                      | 2. Only one graph per problem can be plotted.                           |
|                   |                      |                                                       |                                      |                                                                          |
| Positive Clipper  |                      | Y = CLIPP(X)                                          | 316                                  | It clips off the positive portion of the input waveform. Input X must be floating-point variable. |
|                   |                      | Y = 0 for X ≥ 0.                                      |                                      |                                                                          |
|                   |                      | Y = X for X &lt; 0.                                      |                                      |                                                                          |
| Pulse Generator   |                      | Output is a series of pulses of fixed pulse width Δt. | 1428                                 | 1. The pulse width is fixed at one integration period.                  |
|                   |                      | Pulse amplitude is H, pulse period is TP.             |                                      | 2. TP is the period, a floating-point constant.                         |
|                   |                      |                                                       |                                      | 3. H is pulse amplitude, a floating-point constant.                      |
|                   |                      |                                                       |                                      | 4. A maximum of 5 pulse generators may be used in a problem.             |</p>
<table>
<thead>
<tr>
<th>Name-Symbol</th>
<th>Patching Instruction</th>
<th>Output (typical)</th>
<th>Core Storage requirements FORTRAN II</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Punch-Output</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Symbol</td>
<td>CALL PHOUT(N, M, A, B, C, D, E)</td>
<td>Output data is punched on a card. A maximum of 5 numbers per card.</td>
<td>1718</td>
<td>1. The punch format is F15.5.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2. A maximum of 5 values can be punched out by one calling statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3. Dummy variables are required to replace the unused inputs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4. N, a designator, indicates the number of values to be punched in a single calling statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5. M, a designator, indicates the punching period.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6. Both M and N are fixed-point constants.</td>
</tr>
<tr>
<td>Random-Noise Generator</td>
<td>Y = RANDOM(N)</td>
<td>A sequence of random numbers whose magnitudes are bounded between ± 1.</td>
<td>534</td>
<td>N is a dummy fixed-point constant.</td>
</tr>
<tr>
<td>RNG -&gt; Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name-Symbol</td>
<td>Patching Instruction</td>
<td>Output (typical)</td>
<td>Core Storage requirements</td>
<td>Remarks</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------</td>
<td>------------------</td>
<td>---------------------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| Relay Y = RELAY(X1, X2, C) | Y = X1 for C ≤ 0  
Y = X2 for C > 0 |                  | C is the switching signal.  
X1, X2, and C are floating-point variables or constants. |         |
| Sawtooth Generator Y = STOOTH(NT, H) | Output is a series of sawtooth wave. |                  | 1. The period is expressed in terms of an integral multiple of integration period, i.e., Period T = NT*Δt.  
NT is an integer.  
2. H is the amplitude. It must be a floating-point constant.  
3. A maximum of 5 sawtooth generator may be used. |         |
<p>| Square-wave Generator Y = SQUARE(PT, H) | Output is a series of square-wave. |                  | PT is the period, H is the amplitude. Both are floating-point constants. A maximum of 5 square wave generators may be used in a program. |         |</p>
<table>
<thead>
<tr>
<th>Name-Symbol</th>
<th>Patching Instruction</th>
<th>Output (typical)</th>
<th>Core Storage requirements FORTAN II</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Track-Transfer</td>
<td></td>
<td>Y = TRACK(X1, X2, A, B)</td>
<td></td>
<td>1. A and B are logical signals which control the mode of operation. They are floating-point variables or constants.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EM = X1 for A=1</td>
<td></td>
<td>2. X1, X2 are floating-point variables.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EM = previous value of X1 (hold) for A=0</td>
<td></td>
<td>3. A maximum of 10 Track-transfers may be used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y = X2 for B=1</td>
<td></td>
<td>4. EM is the address of the virtual memory of this element.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y = EM for B=0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type-Output</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Symbol</td>
<td>CALL TYOUT(N, A, B, C, D, E)</td>
<td>Output data is typed out by the computer. A maximum of 5 values per line.</td>
<td>1400</td>
<td>N, a designator, indicates the number of values to be typed out in a single calling statement. N must be an integer not greater than 5. The type format is F15.5.</td>
</tr>
<tr>
<td>Zero-order Hold</td>
<td>Y = ZHOLD(X, SP)</td>
<td>Samples whenever SP &gt;0.</td>
<td>1274</td>
<td>X is the input signal to be sampled. SP is the sampler pulse train. Both are floating-point variables. A maximum of 10 zero order hold may be used in a problem.</td>
</tr>
<tr>
<td>Name-Symbol</td>
<td>Patching Instruction</td>
<td>Output (typical)</td>
<td>Core Storage requirements FORTRAN II</td>
<td>Remarks</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------</td>
<td>------------------</td>
<td>--------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ARCTANGENT</td>
<td>Y = ATANF(X)</td>
<td>Y = arctan(X)</td>
<td>not available</td>
<td>This is one of the functions in the FORTRAN subroutine deck.</td>
</tr>
<tr>
<td>COSINE</td>
<td>Y = COSF(X)</td>
<td>Y = cos(X)</td>
<td>not available</td>
<td>This is the FORTRAN mathematical function.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Angle X given in radians.</td>
</tr>
<tr>
<td>EXPONENTIAL</td>
<td>Y = EXPF(X)</td>
<td>Y = exp(X)</td>
<td>not available</td>
<td>This is the FORTRAN mathematical function.</td>
</tr>
<tr>
<td>GAIN</td>
<td>Y = P1*X</td>
<td>Y = P1X</td>
<td>not available</td>
<td>This is an ordinary FORTRAN arithmetic statement.</td>
</tr>
<tr>
<td>SQUARE ROOT</td>
<td>Y = SQRTF(X)</td>
<td>Y = \sqrt{X}</td>
<td>not available</td>
<td>This is the FORTRAN mathematical function.</td>
</tr>
<tr>
<td>Name-Symbol</td>
<td>Patching Instruction</td>
<td>Output (typical)</td>
<td>Core Storage requirements FORTRAN II</td>
<td>Remarks</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------</td>
<td>------------------</td>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CONSTANT</td>
<td></td>
<td></td>
<td></td>
<td>This is an ordinary FORTRAN arithmetic statement.</td>
</tr>
<tr>
<td>1.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y = P1</td>
<td></td>
<td>Y = P1</td>
<td>not available</td>
<td></td>
</tr>
<tr>
<td>ABSOLUTE VALUE</td>
<td></td>
<td></td>
<td></td>
<td>This is the FORTRAN mathematical function.</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y = ABSF(X)</td>
<td></td>
<td>Y = ABSOLUTE</td>
<td>not available</td>
<td></td>
</tr>
<tr>
<td>VALUE OF X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SINE</td>
<td></td>
<td></td>
<td></td>
<td>This is the FORTRAN mathematical function. The angle X given in radians.</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y = SINF(X)</td>
<td></td>
<td>Y = sin(X)</td>
<td>not available</td>
<td></td>
</tr>
<tr>
<td>SUMMER</td>
<td></td>
<td></td>
<td></td>
<td>This is an ordinary FORTRAN arithmetic statement.</td>
</tr>
<tr>
<td>X1 X2 X3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y = X1+X2+X3</td>
<td></td>
<td>Y = X1+X2+X3</td>
<td>not available</td>
<td></td>
</tr>
<tr>
<td>Name-Symbol</td>
<td>Patching Instruction</td>
<td>Output (typical)</td>
<td>Core Storage requirements FORTRAN II</td>
<td>Remarks</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
<td>------------------</td>
<td>--------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>MULTIPLIER</td>
<td>$Y = X_1 \times X_2$</td>
<td>$Y = X_1X_2$</td>
<td>not available</td>
<td>This is an ordinary FORTRAN arithmetic statement.</td>
</tr>
<tr>
<td>DIVIDER</td>
<td>$Y = \frac{X_1}{X_2}$</td>
<td>$Y = X_1/X_2$</td>
<td>not available</td>
<td>This is an ordinary FORTRAN arithmetic statement.</td>
</tr>
<tr>
<td>DUMMY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBROUTINE SIDAC$n$</td>
<td>none</td>
<td>none</td>
<td>472</td>
<td>$n = 1, 2$ and $3$. They are used to replace any of the unused SUBROUTINE SIDAC$n$.</td>
</tr>
<tr>
<td>MAIN CONTROL</td>
<td>none</td>
<td>none</td>
<td>17746</td>
<td>See block diagram and program listing in Appendix D.</td>
</tr>
</tbody>
</table>
APPENDIX C

C SIDAC-I CODING, TESTING SIDAC-I FUNCTION
C GENERATOR

SUBROUTINE SIDACI(I, A, B, C, D)
T = CLOCK(1.0)
Y1 = FUNG(T, 1, 1)
Y2 = FUNG(T, 2, 2)
Y3 = SINF(3.1416/10.0*T)
ER1 = ABSF(Y3-Y1)
ER2 = ABSF(Y3-Y2)
CALL PHOUT(5, 4, Y3, Y1, Y2, ER1, ER2)
RETURN
END

* DATA

<table>
<thead>
<tr>
<th></th>
<th>0.1</th>
<th>1.0</th>
<th>20.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>1.11</td>
<td>0.34202</td>
</tr>
<tr>
<td>6.13</td>
<td>0.93969</td>
<td>7.8</td>
<td>0.70711</td>
</tr>
<tr>
<td>11.60</td>
<td>-0.5</td>
<td>14.7</td>
<td>-0.99756</td>
</tr>
<tr>
<td>20.0</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>1.11</td>
<td>0.34202</td>
</tr>
<tr>
<td>6.13</td>
<td>0.93969</td>
<td>7.8</td>
<td>0.70711</td>
</tr>
<tr>
<td>11.60</td>
<td>-0.5</td>
<td>14.7</td>
<td>-0.99756</td>
</tr>
<tr>
<td>20.0</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C SIDAC-I CODING, TESTING SIDAC-I INTEGRATORS

SUBROUTINE SIDAC1(I, X, D, D, D)

T = CLOCK(1.0)
DX = SINT(-X)
X = SINT(DX)
SQDX = DX*DX
SQX = X*X
SUM = SQDX+SQX
ERROR = ABSF(SUM-1.0)
CALL TYOUT(5, T, DX, X, SUM, ERROR)
RETURN
END

* DATA

   0. 1   2. 0   20. 0
   IC    2
   0. 0   1. 0
Block Diagrams and FORTRAN Programs of the SIDAC-I Elements

C SIDAC-I MAIN CONTROL, PLOT-VERSION
DIMENSION GX1(20),GX2(20),GX3(20),GY1(20),GY2(20),GY3(20),Y(15),P(20),HEAD(18)
COMMON T,ISKIP,ICT,NF,IZ,NTM,DT,NSQ,NST,NP,KK,1NPER,IR,IPAS,IMC,ID,IS,Y,RESET,JZ,PAR1,P,NP1,N
JP2,NP3,GX1,GX2,GX3,GY1,GY2,GY3,JP1=1,JP2=0,JP3=0
1 TYPE 2
2 FORMAT(10X,52HSIDAC-I A SIMULATED DIGITAL-ANAL
10G (HYBRID) COMPUTER,/) GO TO 23
201 READ 3,DT,PT,FT
3 FORMAT(10X,3F10.0)
4 FORMAT(20X,15HTIME INCREMENT=F10.5,/) NPER=PT/DT
NSAMP=FT/PT
RESET=0.0
JZ=0
DO 5 I=1,15 P(I)=0.0
5 Y(I)=0.0
DO 50 I=16,20
50 P(I)=0.0
PAR1=0.0
6 READ 7,LETT,NN
7 FORMAT(10X,A2,8X,I2)
LETT=LETT/10
IF (LETT-4943) 10,8,10
8 READ 9,(Y(I),I=1,NN)
9 FORMAT(7F10.0)
GO TO 6
10 IF (LETT-5741) 12,11,12
11 READ 9,(P(I),I=1,NN)
GO TO 6
12 IF (NN-1) 16,14,16
13 READ 15,JJ,(GX1(I),GY1(I),I=1,JJ)
15 FORMAT(12/(6F10.0)) NP1=JJ
GO TO 6
16 IF (NN-2) 18,17,18
17 READ 15,JJ,(GX2(I),GY2(I),I=1,JJ)
NP2=JJ
GO TO 6
18 IF (NN-3) 181,180,181
180 READ 15, JJ, (GX3(I), GY3(I), I=1-JJ)
    NP3=JJ
GO TO 6
19 IF (LETT=4844) 191, 190, 191
190 NCARD=1
1901 READ 192, (HEAD(I), I=1, 18)
192 FORMAT(18A4)
    TYPE 192, (HEAD(I), I=1, 18)
    NCARD=NCARD+1
    IF (NCARD=NN) 1901, 1901, 6
191 CONTINUE
    IR=7243
    T=0.O
    ISKIP=0
    KK=0.0
    IF (SENSE SWITCH 3) 25, 26
25 JP4=1
    GO TO 1090
26 JP4=0
1090 CONTINUE
    ICT=0
    NF=0
    NTM=0
    NSQ=0
    NST=0
    NP=0
    IZ=0
    IS=0
    IMC=0
    ID=0
    IF (ISKIP) 3002, 3001, 3002
3001 X1=1.0
    X2=1.0
    X3=1.0
    X4=1.0
    GO TO 3003
3002 JP3=0
    JP1=1
3003 CONTINUE
    IF (NUM=1) 2005, 2004, 2005
2004 CALL SIDAC1(ISKIP, X1, X2, X3, X4)
    GO TO 2009
2006 CALL SIDAC2(ISKIP, X1, X2, X3, X4)
    GO TO 2009
2007 IF (NUM=3) 181, 2008, 181
2008 CALL SIDAC3(ISKIP, X1, X2, X3, X4)
2009 CONTINUE
IF (ISKIP) 1091,1091,1092
1091 CONTINUE
    ISKIP=1
    KK=NPER
    PAR1=1.0
    GO TO 1090
1092 CONTINUE
    IF (RESET) 1,1911,1911
1911 CONTINUE
    DTT=DT*0.5
    JP1=0
    DO 22 MY=1,NSAMP
         KK=0
    DO 21 MZ=1,NPER
         IPAS=0
1097   T=T+DTT
    ISKIP=0
1093 CONTINUE
    ICT=0
    NF=0
    NTM=0
    NSQ=0
    NST=0
    NP=0
    IZ=0
    IS=0
    IMC=0
    ID=0
    IF (NUM=1) 2011,2010,2011
2010 CALL SIDAC1(ISKIP,X1,X2,X3,X4)
    GO TO 2015
2011 IF (NUM=2) 2013,2012,2013
2012 CALL SIDAC2(ISKIP,X1,X2,X3,X4)
    GO TO 2015
2013 IF (NUM=3) 181,2014,181
2014 CALL SIDAC3(ISKIP,X1,X2,X3,X4)
2015 CONTINUE
    IF (ISKIP) 1095,1094,1095
1094 CONTINUE
    ISKIP=1
    IF (IPAS) 1093,1093,1099
1099   KK=KK+1
    JP2=0
    GO TO 1093
1095 IF (IPAS) 1096,1096,1098
1096   IPAS=1
    JP2=1
    GO TO 1097
1098 IF (RESET) 1,194,194
194 CONTINUE
   IF (SENSE SWITCH 4) 122,21
21 CONTINUE
22 CONTINUE
122 PAUSE
   IF (SENSE SWITCH 2) 1220,1224
1220 TYPE 24
   24 FORMAT(41HTURN ON SENSE SWITCH 3 FOR NEW PLOT 1FRAME)
       JP3=1
   TYPE 1221
1221 FORMAT(17HENTER NEW PROGRAM)
   23 TYPE 1222
1222 FORMAT(19H( ) PROGRAM NUMBER)
       ACCEPT 1223,NUM
1223 FORMAT(1X,I2)
       GO TO 201
1224 CONTINUE
   IF (SENSE SWITCH 1) 30,131
30 TYPE 24
       JP3=1
       GO TO 1
131 IF (SENSE SWITCH 4) 132,21
132 READ 3,DT,PT,FT
       NPER=PT/DT
       NSAMP=FT/PT
       GO TO 1911
181 TYPE 182
182 FORMAT (13HSIDAC ERROR 1)
       PAUSE
       GO TO 1
.END
C SIDAC-I MAIN CONTROL, NO PLOT-VERSION
DIMENSION GX1(20),GX2(20),GX3(20),GY1(20),GY2(20),GY3(20),P(20),HEAD(18)
COMMON T,ISKIP,ICT,NF,IZ,NTM,DT,NSQ,NST,NP,KK,
IH,IPAS,IMC,IS,Y,RESET,JZ,PAR1,P,NP1,N
2P2,NP3,GX1,GX2,GX3,GY1+GY2,GY3
1 TYPE 2
2 FORMAT(10X,5HSIDAC-I A SIMULATED DIGITAL-ANAL
10G (HYBRID) COMPUTER,//)
GO TO 23
201 READ 3,DT,PT,FT
3 TYPE 4,DT
4 FORMAT(10X,15HTIME INCREMENT=F10.5,///)
NPER=PT/DT
NSAMP=FT/PT
RESET=0.0
JZ=0
DO 5 I=1,15
P(I)=0.0
5 Y(I)=0.0
DO 50 I=16,20
50 P(I)=0.0
PAR1=0.0
6 READ 7,LETT,NN
7 FORMAT(10X,A2,8X,I2)
   LETT=LETT/10
   IF (LETT-4943) 10*8,10
8 READ 9,(Y(I),I=1,NN)
9 FORMAT(7F10.0)
GO TO 6
10 IF(LETT-5741) 12*11,12
11 READ 9,(P(I),I=1,NN)
GO TO 6
12 IF (LETT-6341) 19*13,19
13 IF (NN-1) 16*14,16
14 READ 15,JJ,(GX1(I),GY1(I),I=1,JJ)
15 FORMAT(I2/(6F10.0))
   NP1=JJ
GO TO 6
16 IF (NN-2) 18*17,18
17 READ 15,JJ,(GX2(I),GY2(I),I=1,JJ)
   NP2=JJ
GO TO 6
18 IF (NN-3) 181*180,181
180 READ 15,JJ,(GX3(I),GY3(I),I=1,JJ)
   NP3=JJ
GO TO 6
19 IF (LETT$ \neq 4844) 191, 190, 191
190 NCARD=1
1901 READ 192, (HEAD(I), I=1, 18)
192 FORMAT (18A4)
TYPE 192, (HEAD(I), I=1, 18)
NCARD=NCARD+1
IF (NCARD=NN) 1901, 1901, 6
191 CONTINUE
IR=7243
T=0.0
ISKIP=0
KK=0.0
1090 CONTINUE
ICT=0
NF=0
NTM=0
NSQ=0
NST=0
NP=0
IZ=0
IS=0
IMC=0
ID=0
IF (ISKIP) 3002, 3001, 3002
3001 X1=1.0
X2=1.0
X3=1.0
X4=1.0
3002 CONTINUE
IF (NUM=1) 2005, 2004, 2005
2004 CALL SIDAC1(ISKIP, X1, X2, X3, X4)
GO TO 2009
2006 CALL SIDAC2(ISKIP, X1, X2, X3, X4)
GO TO 2009
2007 IF (NUM=3) 181, 2008, 181
2008 CALL SIDAC3(ISKIP, X1, X2, X3, X4)
2009 CONTINUE
IF (ISKIP) 1091, 1091, 1092
1091 CONTINUE
ISKIP=1
KK=NPER
PAR1=1.0
GO TO 1090
1092 CONTINUE
IF (RESET) 1, 1911, 1911
1911 CONTINUE
DTT=DT*0.5
DO 22 MY=1,NSAMP
     KK=0
DO 21 MZ=1,NPER
     IPAS=0
1097     T=T+DTT
     ISKIP=0
1093 CONTINUE
     ICT=0
     NF=0
     NTM=0
     NSQ=0
     NST=0
     NP=0
     IZ=0
     IS=0
     IMC=0
     ID=0
     IF (NUM-1) 2011,2010,2011
2010     CALL SIDAC1(ISKIP,X1,X2,X3,X4)
     GO TO 2015
2012     CALL SIDAC2(ISKIP,X1,X2,X3,X4)
     GO TO 2015
2013 IF (NUM-3) 181,2014,181
2014     CALL SIDAC3(ISKIP,X1,X2,X3,X4)
2015 CONTINUE
     IF (ISKIP) 1095,1094,1095
1094 CONTINUE
     ISKIP=1
     IF (IPAS) 1093,1093,1099
1099     KK=KK+1
     GO TO 1093
1095 IF (IPAS) 1096,1096,1098
1096     IPAS=1
     GO TO 1097
1098 IF (RESET) 1,194,194
194 CONTINUE
     IF (SENSE SWITCH 4) 122, 21
21 CONTINUE
     22 CONTINUE
122 PAUSE
     IF (SENSE SWITCH 2) 1220,1224
1220 TYPE 1221
1221 FORMAT(17HENTER NEW PROGRAM)
     23 TYPE 1222
1222 FORMAT(19H( ) PROGRAM NUMBER)
     ACCEPT 1223,NUM
1223 FORMAT(1X,12)
GO TO 201

1224 CONTINUE
   IF (SENSE SWITCH 1) 1, 131
131 IF (SENSE SWITCH 4) 132, 21
132 READ 3, DT, PT, FT
     NPER=PT/DT
     NSAMP=FT/PT
   GO TO 1911

181 TYPE 182
182 FORMAT (13HSIDAC ERROR 1)
     PAUSE
   GO TO 1

END
AUTOMATIC STOP
SUBROUTINE QUIT(X1,X2)
  IF (X1==X2)  2,2,1
  1 PAUSE
  2 RETURN
END

AND GATE
FUNCTION AND(X1,X2,X3,X4,X5)
  IF (X1)  6,6,1
  1 IF (X2)  6,6,2
  2 IF (X3)  6,6,3
  3 IF (X4)  6,6,4
  4 IF (X5)  6,6,5
  5  AND=1.0
     RETURN
  6  AND=0.0
     RETURN
END

BANG-BANG
FUNCTION BANG(X,P)
  IF (X)  1,2,3
  1  BANG=-P
     RETURN
  2  BANG=0.0
     RETURN
  3  BANG=P
     RETURN
END

CLOCK
FUNCTION CLOCK(A1)
COMMON T
  CLOCK=A1*T
RETURN
END
C
COMPARATOR
FUNCTION COMP(X1,X2)
IF(X1-X2) 2.2,1
1  COMP=1.0
RETURN
2  COMP=0.0
RETURN
END

C
COUNTER
FUNCTION COUNT(X,P)
DIMENSION XIN(10),COU(10),STC(10)
COMMON T,ISKIP,ICT
ICT=ICT+1
IF (ISKIP) 103,103,104
IF (ICT-10) 100,100,101
CONTINUE
IF (T) 1*1,5
1 IF (X) 3*3,4
2  COU(ICT)=0.0
   XIN(ICT)=X
   COUNT=COU(ICT)
   STC(ICT)=COUNT
RETURN
4  COU(ICT)=1.0
   GO TO 2
5  IF (X) 2*2,7
6  IF (XIN(ICT)) 9*9,2
9  NP=P
   PN=NP
   IF (COU(ICT)-PN) 10,11,11
10  COU(ICT)=COU(ICT)+1.0
   GO TO 2
11  COU(ICT)=1.0
   GO TO 2
104  COUNT=STC(QUIT)
RETURN
101 TYPE 102
102 FORMAT (14HSIDAC ERROR 13)
PAUSE
END
C
DEAD SPACE
FUNCTION DEADSP(X,UB,AB)
  IF (X) 3,4,1
  1   DIFF=X-UB
  IF (DIFF) 4,4,2
  2   DEADSP=DIFF
  RETURN
  3   DIFF=X-AB
  IF (DIFF) 2,4,4
  4   DEADSP=0.0
  RETURN
END

C
DELAY
FUNCTION DELAY(X)
DIMENSION DEL(20),OUT(20)
COMMON T,ISKIP,ICT,NF,IZ,NTM,DT,NSQ,NST,NP,KK,
1NPER,IR,IPAS,IMC,ID
    ID=ID+1
  7   IF (ISKIP) 7,7,8
  7   IF (ID-20) 1,1,5
  1   IF (T) 3,3,2
  2   OUT(ID)=DEL(ID)
    GO TO 4
  3   OUT(ID)=0.0
  4   DEL(ID)=X
  8   DELAY=OUT(ID)
  RETURN
  5   TYPE 6
  6   FORMAT(14H$SIDAC ERROR 14)
    PAUSE
END

C
FLIP-FLOP
FUNCTION FF(R,S,TR)
DIMENSION FST(15),STF(15)
COMMON T,ISKIP,ICT,NF
    NF=NF+1
  13  CONTINUE
    IF (ISKIP) 13,13,14
  11  CONTINUE
    IF (NF-15) 11,11,10
  1   IF (T) 1,5,1
  1   IF (TR) 2,2,4
2 IF (S) 3,3,8
3 IF (R) 9,9,7
4 IF (FST(NF)) 8,8,7
5 IF (TR) 6,6,8
6 IF (S) 7,7,8
7 FST(NF) = 0.0
  GO TO 9
8 FST(NF) = 1.0
9 FF = FST(NF)
  STF(NF) = FF
  RETURN
14 FF = STF(NF)
  RETURN
10 TYPE 12
12 FORMAT (13HSIDAC ERROR 5)
  PAUSE
END

FUNCTION GENERATOR
FUNCTION FUNG(X,NTAB,NOR)
DIMENSION GX1(20),GX2(20),GX3(20),GY1(20),GY2(120),GY3(20),Y1(15),P(20),FY(20),FX(20),STFG(3)
COMMON T,SKIP,ICT,NF,IZ,NTM,DT,NSQ,NST,NSD,KK,
  INPER,IR,IPAS,IMC,ID,IS,Y,RESET,JZ,PAR1,P,NP1,N
2P2*NP3*GX1*GX2*GX3*GY1*GY2*GY3
IF (ISKIP) 110,100,110
100 IF (NTAB=1) 3,1,3
  N=NP1
  DO 2 I=1,N
  FY(I)=GY1(I)
  2 FX(I)=GX1(I)
  GO TO 9
3 IF (NTAB=2) 6,4,6
  N=NP2
  DO 5 I=1,N
  FY(I)=GY2(I)
  5 FX(I)=GX2(I)
  GO TO 9
6 IF (NTAB=3) 7,7,19
  N=NP3
  DO 8 I=1,N
  FY(I)=GY3(I)
  8 FX(I)=GX3(I)
9 CONTINUE
IF (X) 11.11.12
11 FUNG=FY(1)
   GO TO 21
12 DO 13 I=2,N
   IF (FX(I)-X) 13.14.15
13 CONTINUE
   FUNG=FY(I)
   GO TO 21
15 FUNG=FY(I-1)+(FY(I)-FY(I-1))/(FX(I)-FX(I-1))*(
   1X-FX(I-1))
   STFG(NTAB)=FUNG
   IF (NOR-1) 18.18.16
16 IF (I-N+1) 17.17.18
17 FUNG=FUNG+(X-FX(I-1))*(X-FX(I))/(FX(I+1)-FX(I-1))*
   ((FY(I+1)-FY(I))/(FX(I+1)-FX(I))-(FY(I)-FY
   2(I-1))/(FX(I)-FX(I-1)))
   STFG(NTAB)=FUNG
18 RETURN
110 FUNG=STFG(NTAB)
   RETURN
19 TYPE 20
20 FORMAT(13HSIDAC ERROR 7)
   PAUSE
   END

C

FUNCTION SLIMIT(X,P1,P2)
   DIFF=X-P1
   IF (DIFF) 2.1.1
   1 SLIMIT=P1
      RETURN
   2 DIFF=X-P2
   IF (DIFF) 3.4.4
   3 SLIMIT=P2
      RETURN
   4 SLIMIT=X
      RETURN
   END
MODE CONTROLLED INTEGRATOR

FUNCTION CINT(EIC, P1, P2, X)
DIMENSION YY(20), YK(20), DYDT(20), GATE(20)
COMMON T, ISKIP, ICT, NF, IZ, NTM, DT, NSQ, NST, NP, KK,
1NPERR, IR, IPAS, IMC

IMC = IMC + 1
IF (IMC = 20) 1, 1, 21
1 IF (T) 2, 2, 26
2 GATE(IMC) = 0.0
IF (ISKIP) 3, 3, 5
3 YY(IMC) = EIC
4 CINT = YY(IMC)
RETURN
5 DYDT(IMC) = X
GO TO 4
26 IF (GATE(IMC)) 17, 6, 24
6 IF (IPAS) 7, 7, 10
7 IF (ISKIP) 8, 8, 5
8 IF (P1) 12, 12, 9
9 YK(IMC) = YY(IMC)
YY(IMC) = YY(IMC) + 0.5 * DT * DYDT(IMC)
GO TO 4
10 IF (ISKIP) 11, 11, 5
11 YY(IMC) = YK(IMC) + DT * DYDT(IMC)
GO TO 4
12 IF (P2) 17, 17, 13
13 DYDT(IMC) = 0.0
24 IF (IPAS) 14, 14, 15
14 GATE(IMC) = 1.0
GO TO 3
15 IF (ISKIP) 14, 14, 16
16 GATE(IMC) = 0.0
GO TO 3
17 IF (IPAS) 18, 18, 19
18 GATE(IMC) = -1.0
GO TO 5
19 IF (ISKIP) 18, 18, 20
20 GATE(IMC) = 0.0
GO TO 5
21 TYPE 22
22 FORMAT(14HSIDAC ERROR 11)
PAUSE
END
C NAND GATE
FUNCTION NAND(X1,X2,X3,X4,X5)
  IF (X1) 6,6,1
  1 IF (X2) 6,6,2
  2 IF (X3) 6,6,3
  3 IF (X4) 6,6,4
  4 IF (X5) 6,6,5
  5 NAND=0
   RETURN
  6 NAND=1
   RETURN
END

C NEGATIVE CLIPPER
FUNCTION CLIPN(X)
  IF (X) 1,1,2
  1 CLIPN=0
   RETURN
  2 CLIPN=X
   RETURN
END

C NOR GATE
FUNCTION NOR(X1,X2,X3,X4,X5)
  IF (X1) 1,1,6
  1 IF (X2) 2,2,6
  2 IF (X3) 3,3,6
  3 IF (X4) 4,4,6
  4 IF (X5) 5,5,6
  5 NOR=1
   RETURN
  6 NOR=0
   RETURN
END

C NOT GATE
FUNCTION NOT(X)
  IF (X) 2,2,1
  1 NOT=0
   RETURN
  2 NOT=1
   RETURN
END
C
FUNCTION OFFSET(X,P)
OFFSET=X+P
RETURN
END

C
FUNCTION OR(X1,X2,X3,X4,X5)
IF (X1) 1,1,6
1 IF (X2) 2,2,6
2 IF (X3) 3,3,6
3 IF (X4) 4,4,6
4 IF (X5) 5,5,6
5 OR=0,0
RETURN
6 OR=1,0
RETURN
END

C
PARAMETER
SUBROUTINE PAR(N,A1,A2,A3,A4,A5)
DIMENSION P(20),Y(15)
COMMON T,ISKIP,ICT,NF,IZ,NTM,DT,NSQ,NST,NP,KK,
INPER,IR,IPAS,IMC,ID,IS,Y,RESET,JZ,PAR1,P
IF (PAR1) 11,1,13
1 JZ=JZ+1
2 IF (JZ=20) 2,2,11
3 A1=P(JZ)
4 IF (N=1) 11,13,3
5 JZ=JZ+1
6 IF (JZ=20) 4,4,11
7 A2=P(JZ)
8 IF (N=2) 11,13,5
9 JZ=JZ+1
10 IF (JZ=20) 6,6,11
11 A3=P(JZ)
12 IF (N=3) 11,13,7
13 JZ=JZ+1
14 IF (JZ=20) 8,8,11
15 A4=P(JZ)
16 IF (N=4) 11,13,9
17 JZ=JZ+1
18 IF (JZ=20) 10,10,11
19 A5=P(JZ)
20 IF (N=5) 11,13,11
11 TYPE 12
12 FORMAT(13H$D12 ERROR 2)
   A1=1.0
   A2=1.0
   A3=1.0
   A4=1.0
   A5=1.0
   PAUSE
       RESET= -1.0
   CONTINUE
   RETURN
   END

SUBROUTINE PLOTER(X,Y)
   COMMON T,ISKIP,ICT,NF,I2,NTM,DT,NSQ,NST,NS,NK
   1NPER,IR,IPAS,JP1,JP2,JP3,JP4
   IF (JP3) 1,1,7
   1 IF (JP1) 4,4,2
   2 IF (ISKIP) 3,3,5
   3 CALL PLOT(1:-100.100.10.,100.-100..100.100..10..100.)
       CALL PLOT(99)
       GO TO 6
   4 IF (ISKIP) 10.10.6
   10 IF (JP2) 6.6.5
      XC=X
      YC=Y
       CALL PLOT(90,XC,YC)
      RETURN
   6 CALL PLOT(99)
       IF (JP4) 1,1,8
   8 CALL PLOT(7)
       CALL PLOT(1:-100.100.10.,100.-100..100.100..10..100.)
       CALL PLOT(99)
       GO TO 1
   END

C POSITIVE CLIPPER
FUNCTION CLIPP(X)
   IF (X) 1,2,2
   1 CLIPP=X
   RETURN
   2 CLIPP=0.0
   RETURN
   END
C PULSE GENERATOR
FUNCTION PULSE(TN,H)
DIMENSION PB(5),STP(5)
COMMON T,ISKIP,ICT,NF,IZ,NTM,DT,NSQ,NST,NP
NP=NP+1
IF (ISKIP) 103,103,104
103 IF (NP-5) 100,100,101
100 CONTINUE
1 IF (T) 2.1.2
2 1 PB(NP)=-TN PULSE=H
GO TO 5
2 PB(NP)=PB(NP)+DT/2.0
3 IF (PB(NP)) 4.3.3
3 PB(NP)=-TN PULSE=H
GO TO 5
4 PULSE=0.0
5 STP(NP)=PULSE
RETURN
104 PULSE=STP(NP)
RETURN
101 TYPE 102
102 FORMAT (13HSIDAC ERROR 8)
PAUSE
END

C PUNCH OUTPUT
SUBROUTINE PHOUT(N,M,OP1,OP2,OP3,OP4,OP5)
COMMON T,ISKIP
IF (T) 1.1.4
1 IF (ISKIP) 3.3.2
2 KTK=0
GO TO 11
3 RETURN
4 KTK=KTK+1
5 IF (KTK-4*M) 3.5.5
5 KTK=0
GO TO 11
11 IF (N-1) 12.16.12
12 IF (N-2) 13.19.13
13 IF (N-3) 14.20.14
14 IF (N-4) 15.21.15
15 IF (N-5) 23.22.23
16 PUNCH 17,0P1
17 FORMAT(5F15.5)
18 RETURN
19 PUNCH 17,0P1,0P2
RETURN
20 PUNCH 17,0P1,0P2,0P3
RETURN
21 PUNCH 17,0P1,0P2,0P3,0P4
RETURN
22 PUNCH 17,0P1,0P2,0P3,0P4,0P5
RETURN
23 TYPE 24
24 FORMAT(13HSIDAC ERROR 4)
PAUSE
END

C RANDOM NOISE GENERATOR
FUNCTION RANDOM(N)
COMMON T,ISKIP,ICT,NF,IZ,NTM,DT,NSQ,NST,NP,KK,
1NPER,IR,IPAS
IF (T) 2*2*1
1 IF (IPAS) 2*2*4
2 IF (ISKIP) 3*3*4
3 IR=IR*237
FIR=IR
X=FIR/50000.0-1.0
4 RETURN
END

C RELAY
FUNCTION RELAY(X1,X2,P)
IF (P) 2*2*1
1 RETURN RELAY=X1
2 RETURN RELAY=X2
END
C
SAWTOOTH GENERATOR
FUNCTION STOOTH(NT,H)
DIMENSION PC(5),ST(5)
COMMON T,ISKIP,ICT,NF,IZ,NTM,DT,NSQ,NST
ANT=NT
DH=H/(ANT-1.0)
NST=NST+1
IF (ISKIP) 103 103 104
103 IF (NST=5) 100 100 101
100 CONTINUE
IF (T) 2 1 2
1 STOOTH=0.0
PC(NST)= -ANT*DT
ST(NST)=0.0
RETURN
2 PC(NST)=PC(NST)+DT/2.0
IF (PC(NST)) 3 1 1
3 ST(NST)=ST(NST)+DH/2.0
104 CONTINUE
STOOTH=ST(NST)
RETURN
101 TYPE 102
102 FORMAT (13HSIDAC ERROR 9)
PAUSE
END

C
SIMPLE INTEGRATOR
FUNCTION SINT(X)
DIMENSION X(15),DERIV(15),SYY(15)
COMMON T,ISKIP,ICT,NF,IZ,NTM,DT,NSQ,NST,NP,KK,
INPER,IR,IPAS,IMC,ID,IS,Y
IS=IS+1
100 CONTINUE
IF (T) 4 1 4
1 IF (ISKIP) 2 2 3
2 SINT=Y(IS)
RETURN
3 DERIV(IS)=X
GO TO 2
4 IF (IPAS) 7 5 7
5 IF (ISKIP) 6 6 3
6 SYY(IS)=Y(IS)
Y(IS)=Y(IS)+0.5*DT*DERIV(IS)
GO TO 2
7 IF (ISKIP) 8 8 3
8 Y(IS)=SYY(IS)+DT*DERIV(IS)
GO TO 2
101 TYPE 102
102 FORMAT(14HSIDAC ERROR 12)
PAUSE
END

C SQUARE WAVE GENERATOR
FUNCTION SQUARE(PT,H)
DIMENSION PD(5),STSQ(5),IGATE(5)
COMMON T*,ISKIP,ICT,NF*,IZ*,NTM,DT,NSQ
NSQ=NSQ+1
IF (ISKIP) 103,103,104
103 CONTINUE
IF (NSQ=5) 100,100,101
100 CONTINUE
IF (T) 2*1*2
1  IGATE(NSQ)=1
   SQUARE=H
   PD(NSQ) = -PT/2*0
   GO TO 9
2  PD(NSQ)=PD(NSQ)+DT/2*0
   IF (IGATE(NSQ)) 3,3,6
3  IF (PD(NSQ)) 5*4,4
   IGATE(NSQ)=1
   PD(NSQ) = -PT/2*0
   SQUARE=H
   GO TO 9
5  SQUARE = -H
   GO TO 9
6  IF (PD(NSQ)) 8*7,7
7  IGATE(NSQ)=0
   PD(NSQ) = -PT/2*0
   SQUARE = -H
   GO TO 9
8  SQUARE=H
9  STSQ(NSQ)=SQUARE
RETURN
104 SQUARE=STSQ(NSQ)
RETURN
101 TYPE 102
102 FORMAT (14HSIDAC ERROR 10)
PAUSE
END
C TRACK TRANSFER

FUNCTION TRACK(X1,E1,A,B)
DIMENSION EM(10),STT(10)
COMMON T,ISKIP,ICT,NF,IIZ,NTM

NTM=NTM+1

IF (ISKIP) 12,12,13

12 IF (NTM-10) 10,10,9

10 IF (T) 1,1,7

1 IF (A) 3,3,2

2 EM(NTM)=X1

GO TO 4

3 EM(NTM)=0.0

4 IF (B) 6,6,5

5 TRACK=EI

STT(NTM)=TRACK

RETURN

6 TRACK=EM(NTM)

STT(NTM)=TRACK

RETURN

7 IF (A) 4,4,8

8 EM(NTM)=X1

GO TO 4

13 TRACK=STT(NTM)

RETURN

9 TYPE 11

11 FORMAT (13HSIDAC ERROR 6)

PAUSE

END

C TYPE OUTPUT

SUBROUTINE TYOUT(N,OP1,OP2,OP3,OP4,OP5)
COMMON T,ISKIP,ICT,NF,IIZ,NTM,DT,NSQ,NST,NP,KK,
INPER

IF (KK-NPER) 8*1,1

1 IF (N-1) 2*6,2

2 IF (N-2) 3*9,3

3 IF (N-3) 4*10*4

4 IF (N-4) 5*11,5

5 IF (N-5) 13*12,13

6 TYPE 7,OP1

7 FORMAT(5F15.5)

8 RETURN

9 TYPE 7,OP1,OP2

RETURN
10 TYPE 7, OP1, OP2, OP3
   RETURN
11 TYPE 7, OP1, OP2, OP3, OP4
   RETURN
12 TYPE 7, OP1, OP2, OP3, OP4, OP5
   RETURN
13 TYPE 14
14 FORMAT (13H$SIDAC ERROR 3)
   PAUSE
   END

C ZERO ORDER HOLD
FUNCTION ZHOLD(X, SP)
DIMENSION PZH(10), STZ(10)
COMMON T, ISKIP, ICT, NF, IZ
   IZ = IZ + 1
103 IF (ISKIP) 103, 103, 104
103 IF (IZ - 10) 100, 101, 101
100 CONTINUE
   IF (T) 2*1, 2
1 IF (SP) 5*4, 3
3 PZH(IZ) = X
4 ZHOLD = PZH(IZ)
   GO TO 6
5 ZHOLD = 0.0
6 STZ(IZ) = ZHOLD
   RETURN
104 ZHOLD = STZ(IZ)
   RETURN
101 TYPE 102
102 FORMAT (14H$SIDAC ERROR 15)
   PAUSE
   END
Figure D-1. Block diagram of the SIDAC-I Main Control.
RESET all the COUNTERS, i.e., ICT, NF, etc.

ISKIP = 0

T

X1 = 1.0
X2 = 1.0
X3 = 1.0
X4 = 1.0

JP3 = 0
JP1 = 1

NUM = n

CALL SIDACn (ISKIP, ---)

ISKIP > 0

ISKIP = 1
KK = NPER
PAR 1 = 1.0

DTT = DT + 0.5
JP1 = 0

Figure D-1. (continued)
Figure D-1. (continued)
AND (X1, X2)

Figure D-2. Block diagram of the SIDAC-I Automatic Stop.

Figure D-3. Block diagram of the SIDAC-I AND Gate.
Figure D-4. Block diagram of the SIDAC-I Bang-Bang.

Figure D-5. (a) Block diagram of SIDAC-I Clock.
   (b) Block diagram of SIDAC-I Comparator.
Figure D-6. Block diagram of SIDAC-I Counter.
Figure D-7. Block diagram of SIDAC-I Dead Space.
Figure D-8. Block diagram of SIDAC-I Delay.
Figure D-9. Block diagram of the SIDAC-1 Flip-Flop.
Figure D-10. Block diagram of the SIDAC-I Function Generator.
Figure D-10. (continued)
Figure D-11. Block diagram of the SIDAC-I Mode Controlled Integrator.
Figure D-11. (continued)
Figure D-12. Block diagram of the SIDAC-I Limiter.

Figure D-13. Block diagram of the SIDAC-I NAND Gate.
Figure D-14. (a) Block diagram of the SIDAC-I Negative-Clipper. (b) Block diagram of the SIDAC-I NOT Gate.

Figure D-15. Block diagram of the SIDAC-I NOR Gate.
Figure D-16. Block diagram of the SIDAC-1 OR Gate.

Figure D-17. (a) Block diagram of the SIDAC-1 Offset.
(b) Block diagram of the SIDAC-1 Positive-Clipper.
Figure D-18. Block diagram of the SIDAC-I PARAMETER.
Figure D-19. Block diagram of the SIDAC-I PLOTER.
Figure D-20. Block diagram of the SIDAC-I Punch-Output.
Figure D-21. Block diagram of the SIDAC-I Pulse Generator.
Figure D-22. Block diagram of the SIDAC-I Random Noise Generator.

Figure D-23. Block diagram of the SIDAC-I Relay.
Figure D-24. Block diagram of the SIDAC-I Simple Integrator.
Figure D-25. Block diagram of the SIDAC-1 Square-Wave Generator.
Figure D-26. Block diagram of the SIDAC-I Sawtooth Generator.
Figure D-27. Block diagram of the SIDAC-I Type-Output.
Figure D-28. Block diagram of the SIDAC-I Track-Transfer.
Figure D-29. Block diagram of the SIDAC-I Zero-Order Hold.