SU-SEL-71-057

A Modular Organization of a Digital Integrating Computer for the Numerical Solution of Differential Equations

by

E. J. Schulz

December 1971

Technical Report No. 3606-6

This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contract N00014-67-A-0112-0044

RADIOSCIENCE LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY · STANFORD, CALIFORNIA

R

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Stanford Electronics Laboratories Stanford University Stanford, California 94305	Unclassified 2b. GROUP

3. REPORT TITLE

A MODULAR ORGANIZATION OF A DIGITAL INTEGRATING COMPUTER FOR THE NUMERICAL SOLUTION OF DIFFERENTIAL EQUATIONS

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Technical Report No. 3606-6, December 1971

5. AUTHOR(S) *(First name, middle initial, last name)*

E. J. Schulz

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1971	83	42

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
N00014-67-A-0112-0044 b. PROJECT NO.	TR No. 3606-6 SEL-71-057
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT Reproduction in whole or in part is permitted for any purpose of the United States Government. This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Joint Services Electronics Program U.S. Army, U.S. Navy, U.S. Air Force

13. ABSTRACT

The automatic solution of differential equations may be accomplished by either modeling the equation on an analog computer or by solving it numerically on a general-purpose computer. Both methods are cumbersome and have the disadvantages of low accuracy and slow speed, respectively. The development of the digital differential analyzer promised a machine with improved accuracy and speed. The difficulty in programming and the reliance on complex switching networks or patch boards brought about by ever-increasing parallelism, however, have prevented the full exploitation of the DDA capabilities.

A modular machine structure employing serial-parallel processing and using incremental integration as its basic algorithm has been developed. The system consists of self-contained modules which may be operated independently or may be combined to solve numerically one or more differential equations. Modularity and serial-parallel processing simplify the communication methods within and between modules to permit automatic programming; the hardware requirements are reduced as in serial processing, but the iteration time cannot exceed a fixed maximum regardless of the problem.

To eliminate some of the masked instabilities inherent in circular number systems, a two-loop number system is presented. An extension of the two-loop system leads to number systems with a hysteresis. Except for the case of multi-bit communication, it is possible to predict the outcome of the integrating cycle sufficiently to permit post-multiplication of the integral increment by a constant or a variable simultaneously with the integrating cycle. This capability considerably reduces the solution time and required hardware. (Continued)

14	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
DIGITAL INTEGRATION NUMERICAL SOLUTIONS INCREMENTAL COMPUTATION						

ABSTRACT (continued)

Combining the machine with a general-purpose computer allows automatic programming and scaling. In this environment, the user-generated program consists only of the differential equations entered in a standard format, declarations of dependent and independent variables, the number of coupled equations to be solved, and some control statements.

SEL-71-057

A MODULAR ORGANIZATION OF A DIGITAL INTEGRATING COMPUTER FOR THE NUMERICAL SOLUTION OF DIFFERENTIAL EQUATIONS

by

E. J. Schulz

December 1971

Technical Report No. 3606-6

This work was supported in part by the
Joint Services Electronics Program
(U.S. Army, U.S. Navy and U.S. Air Force)
under Contract N00014-67-A-0112-0044

Radioscience Laboratory
Stanford Electronics Laboratories
Stanford University Stanford, California

ABSTRACT

The automatic solution of differential equations may be accomplished by either modeling the equation on an analog computer or by solving it numerically on a general-purpose computer. Both methods are cumbersome and have the disadvantages of low accuracy and slow speed, respectively. The development of the digital differential analyzer promised a machine with improved accuracy and speed. The difficulty in programming and the reliance on complex switching networks or patch boards brought about by ever-increasing parallelism, however, have prevented the full exploitation of the DDA capabilities.

A modular machine structure employing serial-parallel processing and using incremental integration as its basic algorithm has been developed. The system consists of self-contained modules which may be operated independently or may be combined to solve numerically one or more differential equations. Modularity and serial-parallel processing simplify the communication methods within and between modules to permit automatic programming; the hardware requirements are reduced as in serial processing, but the iteration time cannot exceed a fixed maximum regardless of the problem.

To eliminate some of the masked instabilities inherent in circular number systems, a two-loop number system is presented. An extension of the two-loop system leads to number systems with a hysteresis. Except for the case of multi-bit communication, it is possible to predict the outcome of the integrating cycle sufficiently to permit post-multiplication of the integral increment by a constant or a variable simultaneously with the integrating cycle. This capability considerably reduces the solution time and required hardware.

Combining the machine with a general-purpose computer allows automatic programming and scaling. In this environment, the user-generated program consists only of the differential equations entered in a standard format, declarations of dependent and independent variables, the number of coupled equations to be solved, and some control statements.

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont.)

ILLUSTRATIONS

ILLUSTRATIONS (Cont)

TABLES

ACKNOWLEDGMENT

I wish to express my sincere appreciation to my research advisor, Professor Allen M. Peterson, for his guidance and suggestions and to Professors E. Davidson and O. Buneman for their thorough reading of this manuscript. I want to acknowledge also the many helpful discussions on automatic programming with my colleague, B. Parasuraman.

Chapter I

INTRODUCTION

A. Numerical Solution of Differential Equations

The quantitative study of physical systems requires the expression of the system characteristics in mathematical form. This expression usually results in some differential equation which, when evaluated, shows behavior corresponding to that of the original system. The equations may be linear, nonlinear, or partial differential equations.

The solution of differential equations requires that we find some function $y = y(x,C)$ such that if the function y is substituted in the differential equation [say, $dy/dx = f(y,x)$] the result is an identity. Since the function y can be found analytically only in a small number of cases, we resort to numerical methods of finding the solution. Numerical solutions require the complete specification of the differential equation (initial conditions and parameters) and therefore are always particular solutions. The numerical solution may be found by either differentiating or integrating, but integration is employed almost exclusively because differentiation involves the generation of the difference between two very small quantities (which ideally tend toward zero) and therefore introduces unnecessary errors.

Numerical integration is achieved by replacing the integrand with some quadrature formula and evaluating this over the required interval of the independent variable. In this process the independent-variable interval is divided into subintervals which are usually of equal lengths.

Traditionally, two basic methods have been available to obtain the numerical solutions of differential equations. The first is to solve the equation on a general-purpose computer, using such numerical techniques as the modified Euler integration, Adam's trapezoidal integration, or summation of the Taylor series. The second is to model the equation on an analog computer. Given that we desire high-solution speeds and accuracies, neither of these methods is ideal. The general-purpose computer is often too slow, and the analog computer simply cannot provide the accuracy.

If, in the system under study, the dependent variables vary only with respect to time or some other single independent variable, we have an ordinary differential equation; if, on the other hand, the dependent variables vary with respect to two or more independent variables, the equations will contain partial derivatives. Because in the analog computer all integration is with respect to time only, these partial differential equations cannot be solved directly. The use of the "generalized integrator," which includes a multiplier, allows in effect integration with respect to a variable other than time, but the multiplier also introduces additional errors and represents additional hardware and cost.

Because the analog computer is a completely parallel machine (it consists of many processors operating simultaneously), its programs must be hard wired for continuous operation. This requires the use of plug boards.

B. Background

The earlier development of the digital differential analyzer (DDA), which is essentially the digital equivalent of the analog integrator, allowed the modification of analog computers. Replacing analog integrators with DDA integrators resulted in systems capable of high-speed solutions and the desired high accuracies; in addition, the independent variable was no longer restricted to time as in the analog integrator.

The first such machine to be built was the MADDIDA (Bartee et al, 1962), developed in 1950. It was considered a low-cost device, employing a magnetic drum memory to allow arbitrary stored interconnections such that any DDA integrator could be connected to any other integrator, including itself. The MADDIDA used binary communication, which requires a single bit and restricts both the independent variable input and the integral output increment to the values of +1 and -1.

Since 1950, the need for higher speed and accuracies has produced many technological improvements. More accurate algorithms were introduced (Yu, 1968; Nilsen, 1968) and, to increase operating speed, subsequent systems had high degrees of parallelism. This latter trend made it practically impossible to retain stored programs, leading to the alternatives of single-purpose computers or patch-board programming.

The TRICE (Transistorized Real Time Incremental Computer - Expandable), developed by Packard Bell Corporation in 1958 (Mitchell, Ruhman, 1958), was such a machine using plug-board programming and parallel processing at a rate of 100,000 iterations/sec.

Past and present DDAs have been designed essentially in the manner of analog computers, and it is this analog approach which, in my opinion, has prevented the development of a truly general DDA machine which can find wide acceptance. Existing DDAs are for the most part "one application" computers, solving the same equations or sets of equations with different initial conditions or parameters. They are used for navigational calculations, for computation of projectile trajectories, or for high-order control-system equations. Any change in programming involves either hardware modifications (such as plug-board reprogramming) or complex time and/or space multiplexing schemes to effect the proper interconnection of the various integrators. As a result, these methods severely limit the application of the machines because they require either a great amount of time and skill on the part of a programmer or enormous amounts of multiplexing hardware. Such disadvantages have acted as strong deterrents to the full exploitation of the inherent capabilities of digital incremental integration.

C. Statement of the Problem

This investigation has sought a new approach to the problem, directed toward the development and organization of a special-purpose machine to solve differential equations numerically. The goal is a high-speed high-accuracy system that will be compact, adaptable, and, above all, easy to use. Although the proposed system has not yet been constructed as hardware, it has been simulated on the Stanford Computation Center IBM 360 Model 67.

A new machine structure, the digital incremental computer (DIC), based on a modular concept, is proposed. Each module is a separately self-contained device that can operate independently or connected to other modules on one or several problems simultaneously. Its structure is such that if the system is employed in conjunction with a general

purpose computer, it will not only be easy to use but in fact will require substantial effort on the part of the programmer to avoid using it. A software package, developed by B. Parasuraman (Schulz, Parasuraman, 1971), will be employed in conjunction with the DIC to accept the problem statement virtually in the form that differential equations are normally written. Additional statements required are the number of equations to be solved, a declaration of the dependent and independent variables, and specification of the range and precision of the desired solution. The software package will generate the program, load the system, and store the solution output for subsequent use or for printout and display.

The system employs serial-parallel processing which, although slower than total parallel processing, does not allow the iteration time to exceed the time required to process all integrals in one module. The solution time of equations that do not require all available modules can be decreased by distributing the various integral functions over several modules. Serial-parallel processing also allows total communication within the modules and restrictive communication between modules without the necessity of resorting to patch boards or extensive time or space multiplexing. Here, "total communication" means that any integral output can be used as the dependent variable input, or as a component thereof, to any or all integrals; this output also can be used as the independent variable input to any two integrals.

Other innovations are the two-loop number system and simultaneous integration and multiplication. It is shown that the two-loop number system eliminates instabilities and oscillations encountered when employing a circular number system. Increasing the size of the two loops while maintaining the total number-range constant results in number systems containing a hysteresis.

Simultaneous integration and multiplication can decrease the total solution time by one half. With the given number system, it is possible to make a partial prediction of the outcome of the integration at the beginning of each integrating cycle. This prediction is sufficient to allow initiation of the multiplication process of the output by either a constant or a function and to complete the multiplication before the integral output is generated.

A simplified method is developed that allows floating-point arithmetic yet requires the storing and recalculating of only a single exponent for each integral. Furthermore, this floating-point method does permit simultaneous integration and post-multiplication.

D. Approach

Chapter II deals with the principles of numerical solutions and, in particular, centers on solutions using digital differential analyzers. This and the consideration of the basic DDA construction parameters introduce the proper background for subsequent chapters.

Chapter III describes the concept of the proposed machine. The design goal is outlined, and the necessary requirements to meet this goal are established. In Chapter IV several number systems are investigated. The two-loop number system is introduced, and an extension of this system leads to a system of overlapping loops. The logical implementation is presented for both the circular and two-loop number systems, as well as for a multi-bit transfer two-loop system.

Chapter V considers the conceptual functional block. Several innovations such as pre- and post-multiplication are incorporated into the basic block, and simultaneous integration and post-multiplication are introduced. It is shown that the outcome of the integral function in single-bit transfer machines can be predicted. In addition, a floating-point arithmetic method is introduced, which requires the storage of only a single exponent for the total functional block.

The multi-module system with serial-parallel processing is presented in Chapter VI. Two basic approaches, "horizontal communication" and "vertical communication," are considered for inter-module communication.

Chapter VII discusses the module; processing, intra-module communication methods, and the memory organization are examined. Chapter VIII outlines the proposed machine. Operating procedures, communication methods within and between modules, and the externally generated function inputs are explained. An example illustrates the programming. Chapter IX summarizes the work and presents conclusions and suggestions for further study.

Chapter II

PRINCIPLES OF DIGITAL DIFFERENTIAL ANALYZERS

A. Principles of Numerical Solution

To solve numerically for the integral of a function $f(x)$, the function is replaced by a formula that approximates $f(x)$ over a small interval of the independent variable x, and the result is integrated over that interval. The equations employed usually require knowledge of the previous values of the integral, the function, and some of its derivatives. The newly calculated value of the integral then can be used as a factor to compute other functions and to repeat the above process. When the integral has been formed over the interval for which the quadrature formula is valid, that formula must be updated by obtaining new values for the function and its derivatives. These methods are well described in the literature (Scarborough, 1966; Cunningham, 1958).

In incremental integration, we do not obtain the whole integral but only the change of the integral during the subinterval. This change then is transmitted to be used as a factor to calculate other functions, or it can be accumulated to yield the whole integral value.

The most commonly used incremental integrating algorithms are rectangular and modified trapezoidal.

1. Rectangular Integration

If, in the Taylor series

$$f(x_{i+1}) = f(x_i) + f'(x_i)\,\Delta x_i + \frac{1}{2!} f''(x_i)\,\Delta x_i^2 + \frac{1}{3!} f'''(x_i)\,\Delta x_i^3 + \ldots \qquad (2.1)$$

we drop all terms on the right-hand side that contain powers of Δx greater than one, we have

$$f(x_{i+1}) = f(x_i) + f'(x_i)\,\Delta x_i \qquad (2.2)$$

where

$$f'(x_i) = \frac{df(x_i)}{dx}$$

$$\Delta x_i = x_i - x_{i-1}$$

Equation (2.2) represents rectangular integration of $f'(x)$ with respect to x. Figure 1 is the graphical representation of this process. The area that is bounded by the curve y, the abscissa, and the ordinates at the end points of the desired finite interval (x_o, x_n) is divided into small rectangles of height y_i and width $\Delta X_i = x_{i+1} - x_i$. If n is made to be very large, which is equivalent to making Δx very small, and if the function y is well behaved, the sum of these rectangles is an approximation to the integral of y with respect to x over the specified interval:

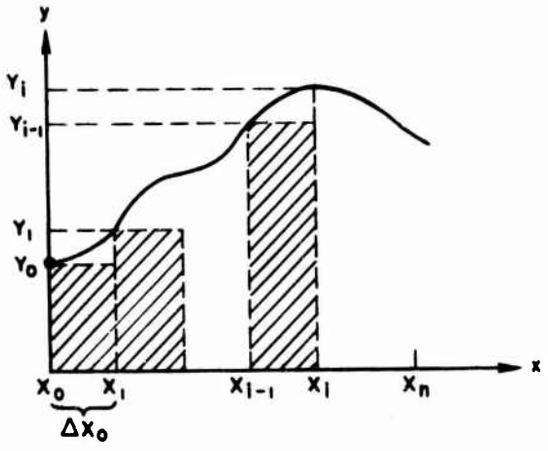


Fig. 1. RECTANGULAR INTEGRATION.

$$\lim_{n \to \infty} \sum_{i=0}^{n} Y_i \Delta X_i = \int_0^{x_n} y \, dx \qquad (2.3)$$

If the integral $\int_0^{x_n} y \, dx = Z$, then the individual $Y_i \Delta X_i$ are the increments ΔZ_i of the integral, and $\Delta Z_i = Y_i \Delta X_i$.

The digital differential analyzer (DDA) is a device that implements this incremental integration. Figure 2 illustrates the basic construction of a DDA, which requires two registers and two arithmetic units, and Fig. 3 is its schematic symbol. The inputs to the DDA are the dependent and independent variable increments ΔY and ΔX, respectively. In the following, all variables are normalized to unity.

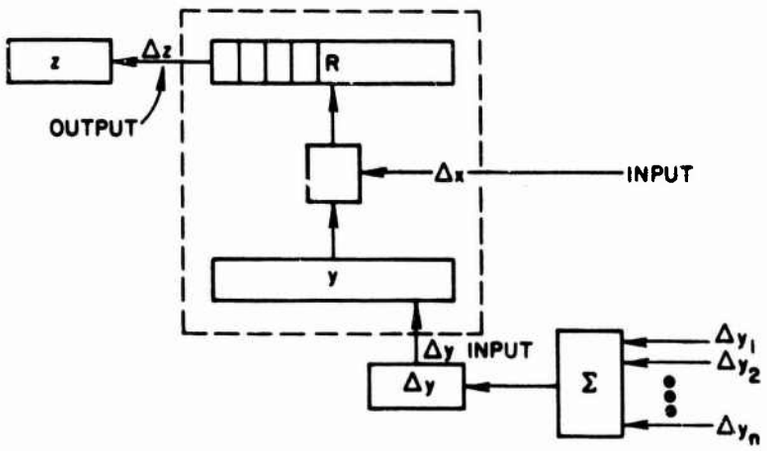


Fig. 2. DIGITAL INTEGRATOR FOR RECTANGULAR INTEGRATION.

Fig. 3. DIGITAL INTEGRATOR SYMBOL.

The value of ΔX is restricted to +1, -1, and 0. The accumulation of the ΔY increments is stored in the Y register and Y is added to the content of the R register if ΔX is positive and subtracted if it is negative. The process then is described by

$$Y_i = \sum_{j=1}^{i} \Delta Y_j + Y_o \tag{2.4}$$

and

$$R_i = \sum_{j=1}^{i} Y_j \Delta X_j + R_o \tag{2.5}$$

where Y_o and R_o are the initial values of the integrand and integral, respectively. These equations can be rewritten as difference equations:

$$Y_i = Y_{i-1} + \Delta Y_i \tag{2.6}$$

and

$$R_i = R_{i-1} + Y_i \Delta X_i \tag{2.7}$$

where R_i is a whole word and is the summation of Y with respect to X. If we consider the maximum allowable absolute value of R to be N, then, whenever $|R_i|$ exceeds N, an overflow or underflow occurs which represents the output ΔZ of the DDA. An accumulation of all ΔZ in some other register will again be equal to the summation of $Y_i \Delta X_i$ with the remainder R_i in the R register:

$$R_i = R_{i-1} + Y_i \Delta X_i - N \Delta Z_i \tag{2.8a}$$

$$R_i = \sum_{j=1}^{i} Y_j \Delta X_j - \sum_{j=1}^{i} N \Delta Z_j \tag{2.8b}$$

2. Modified Trapezoidal Integration

The approximation of the integral can be improved by using trapezoids or some other geometrical areas instead of the elementary rectangle. The most frequently used higher order integrating rule is the modified trapezoidal algorithm; this is an extrapolating algorithm rather than interpolating and is physically realizable for a fast system whereas the interpolating system is not (Yu, 1968). In extrapolating trapezoidal integration, we make a correction to the calculated increment of the integral based on the value of the function derivative during the previous interval. The equations for modified trapezoidal integration are

$$Y_i = Y_{i-1} + \Delta Y_i \tag{2.9}$$

and

$$R_i = R_{i-1} + Y_i \Delta X_i + \frac{1}{2} \Delta Y_i \Delta X_i \tag{2.10}$$

Because the inherent delays in serial processing systems are less than those in parallel systems, the modified trapezoidal integration rule is not a proper algorithm for serial machines. The Rieman integrating rule should be used (Monroe, 1962) for these systems.

B. DDA Solution of Differential Equations

A single DDA solves the differential equation $dz = ydx$. To solve more complex equations, several integrators must be interconnected such that the resulting circuit models the equation. The basic programming techniques are similar to those used for analog computers (Sizer, 1968; Forbes, 1956) and are well known. The equation to be solved is rewritten in differential form with the highest order derivative on the left-hand side and all other terms on the right-hand side. Using the highest order differential term as the dependent-variable increment input, it is integrated with respect to the independent variable. With successive integration, all derivatives of the function can be found, and the terms on the right-hand side of the equation can be generated. The sum of these terms is equal to the highest order differential and the loop is closed. [Other approaches may result in a simpler program for certain problems (Yu, 1968)]. Some examples will be instructive and will serve as a comparison to the program and hardware requirement for the proposed system.

C. Examples

1. Example 1

We will consider the programming of Van der Pol's equation

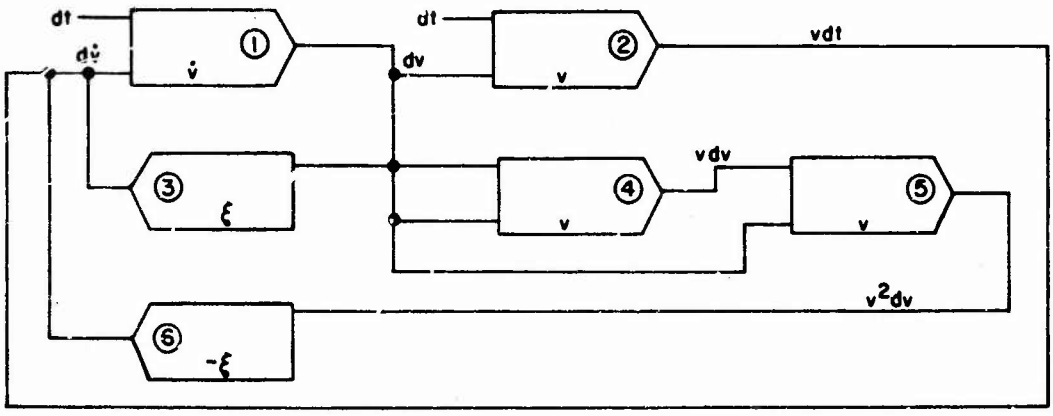
$$\frac{d^2v}{dt^2} - \xi(1 - v^2)\frac{dv}{dt} + v = 0 \qquad (2.11)$$

with the initial conditions $v_o = 1.5$ and $\dot{v}_o = 0$. The maximum value of v and $\dot{v}$ will be less than 2 if ξ is near unity. Multiplying

this equation by dt and moving all terms but the highest order term to the right-hand side yields

$$\dot{dv} = \xi dv - \xi v^2 dv - v dt \qquad (2.12)$$

where $dv = \dot{v}dt$. Letting $\dot{dv}$ be the dependent-variable input to the first integrator and dt its independent-variable input, the output becomes $\dot{v}dt = dv$. Integrating dv with respect to t results in vdt as the output of integrator 2. With the two terms dv and vdt, we can now generate $\dot{dv}$ and close the loop as shown in Fig. 4.



$$\frac{d^2v}{dt^2} - \xi(1-v^2)\frac{dv}{dt} + v = 0$$

Fig. 4. SOLUTION DIAGRAM FOR VAN DER POL'S EQUATION.

The program connection is now determined. To complete the program, we must still scale all variables; several methods are possible. One approach is to solve a set of algebraic equations for each integrator. A single integrator is illustrated in Fig. 5, with the scaling quantities M, N, X, Y, and Z. If the maximum value of the integrand is y_m, then $2^M \geq y_m$. The number of bits used in the integrand register is N; X, Y, and Z are the exponents of 2 such that there are 2^X, 2^Y, and 2^Z increments for each unit of the independent-variable input, dependent-variable input, and integral output, respectively. The integrator is scaled correctly if

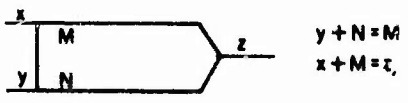


Fig. 5. DIGITAL INTEGRATOR SYMBOL WITH SCALING PARAMETERS.

$$Y + N = M \tag{2.13}$$

and

$$X + M = Z \tag{2.14}$$

All scale factors in the above example are shown in Table 1. In this case, the maximum number of bits was taken to be 16. Note that the independent-variable input for integrator 2 has a scale factor different from that of integrator 1. This situation usually arises if the

Table 1

SCALING FOR $\frac{d^2v}{dt^2} - (1 - v^2)\frac{dv}{dt} + v = 0$

Integrator (No.)	Function (y)	y_m	M	N	X	Y	Z
1	$\dot{v}$	2	1	14	-16	-13	-15
2	v	1.5	1	16	-14	-15	-13
3	ξ	1	2	16	-15	---	-13
4	v	1.5	1	16	-15	-15	-14
5	v	1.5	1	16	-14	-15	-13
6	$-\xi$	-1	0	16	-13	---	-13

original equation is magnitude and frequency scaled in a fashion similar to equations being programmed for analog computers. Unless automatic scaling is available, this method is very often the easiest and simplest because it eliminates the need to solve the several sets of algebraic equations (Peterson, 1968). If the machine does not allow the use of different machine times, the X input for integrator 2 may be generated by an additional integrator with a constant multiplication factor and M = 2.

2. Example 2

A second example is the equation of a circle:

$$y\ddot{y} + \dot{y}^2 + 1 = 0 \qquad (2.15)$$

Assume that we want to solve this equation for values of x from $x = 0$ to $x = 2.1$; then, $y_o = 1$, $\dot{y}_o = 2$, $y_{max} = 2.236$, and $\dot{y}_{max} = 2$. Figure 6 is the connection diagram, and the scaling factors are given in Table 2 (again for a maximum of 16 bits).

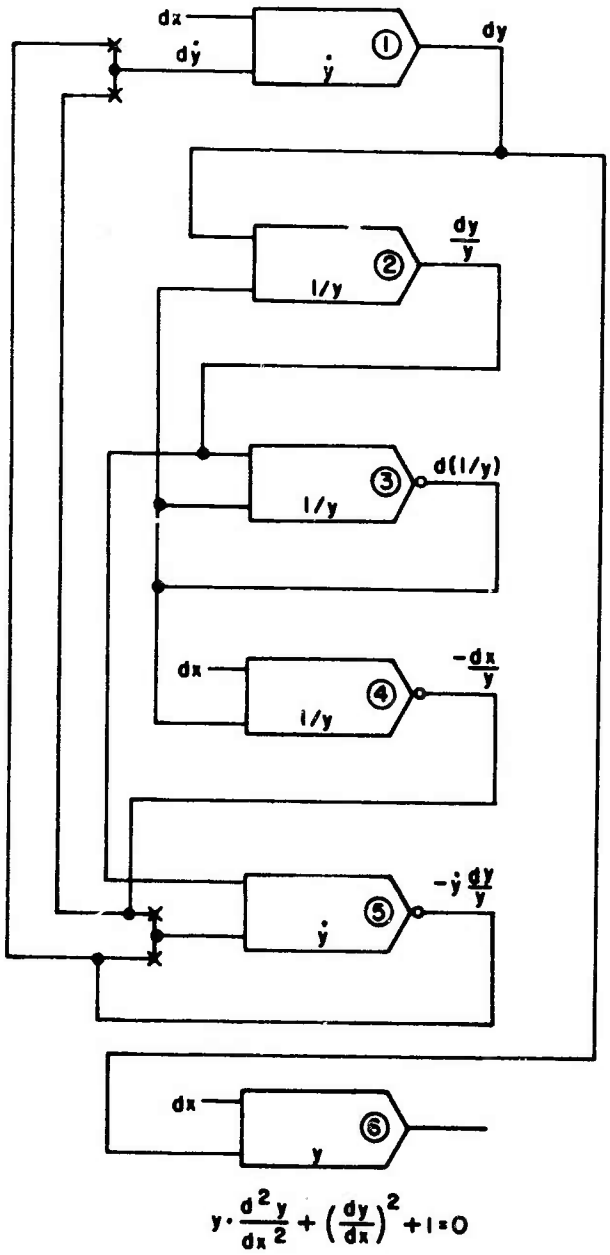


Fig. 6. SOLUTION DIAGRAM FOR $y\ddot{y} + \dot{y}^2 + 1 = 0$.

Table 2

SCALING FOR $y \cdot \frac{d^2y}{dx^2} + \left(\frac{dy}{dx}\right)^2 + 1 = 0$

Integrator (No.)	Function (y)	M	N	X	Y	Z
1	$\dot{y}$	1	14	-15	-13	-14
2	1/y	0	14	-14	-14	-14
3	1/y	0	14	-14	-14	-14
4	1/y	2	16	-15	-14	-13
5	$\dot{y}$	1	14	-14	-13	-13
6	y	2	16	-15	-14	-13

D. Construction Parameters

DDAs can be classified by three basic construction parameters (Wood, 1965):

(1) parallel/serial--input-output

(2) parallel/serial--arithmetic

(3) parallel/serial--processing of integrators

Although they do influence system performance, parameters (1) and (2) chiefly represent possible trade-offs between solution speed and hardware on a fixed-ratio basis. For example, if it takes K_1 clock pulses to process an integrator with parallel arithmetic and it takes K_2 clock pulses to process one integrator of the same bit length with serial arithmetic, then for a given machine (serial or parallel) the solution time using serial arithmetic will be K_2/K_1 multiplied by the solution time using parallel arithmetic regardless of the equation being solved. The third parameter, however, represents trade-offs of variable ratios between solution speed and hardware complexity. Parallel processing results in the highest possible solution speed, and the iteration rate for a given word length is constant regardless of the equation being solved.

A machine that has a fixed number of integrators can solve any problem as long as the availability of integrators is not exhausted. For a practically sized machine, this might constitute a severe limitation on the type of problems that can be solved.

Programs for parallel-processing machines must of necessity be hard wired and the machine, therefore, cannot use stored programs. The usual programming method in this case is either permanently wiring for a machine that solves only one problem with different equation parameters (or initial conditions) or plug-board programming as used on analog computers.

Plug-board programming requires an excessive amount of time compared to the solution time. Although this may be acceptable, if the particular program is a standard program to be used many times, this method is not feasible in a situation where the machine is to be employed by many users with different problems. In addition, wiring errors may easily occur, especially in problems requiring a high wiring density.

The solution time for serial processing DDAs varies directly with the complexity of the problem because only one integrator is processed at any one time. This, however, considerably simplifies the problem of programming. Only one pair of input variables and one output variable must be generated or transmitted with stored programs. Limitations on size or complexity of problems that can be solved on a particular machine are, in this case, set by the size of the available program storage. The machine requires only a single processor.

When programming any DDA, the programmer must be able to manipulate the differential equation to be solved in such a way that he can set up a solution diagram. This often requires recognition of functions as solutions to differential equations which, in turn, are necessary for the overall solution of the given problem. This requirement on the programmer in itself restricts the practical use of DDA machines to a relatively small number of users. One of the most important considerations in this work, therefore, has been to develop a machine environment that would eliminate the most difficult and time-consuming programming tasks.

Chapter III

CONCEPT OF THE PROPOSED MACHINE

A. Requirements

The objective of this work was to develop a machine that would satisfy the following goals. (1) It is to operate in the environment of a computer center mated to either a small or large computer. (2) It must be capable of operating with user-generated programs that specify little more than the equation to be solved, the dependent and independent variables, and the accuracy and range of the solution desired. The mapping of the equation and the generation of the machine program, therefore, must be accomplished automatically. (3) In addition to acting as an external device to a general-purpose computer (GPC), it must be able to operate independently in the environment of control systems; this is important because many control systems are complex enough to require the solution of differential equations but do not warrant the expenditure of a large high-speed general-purpose computer.

The basic requirements selected were

(1) high accuracy
(2) high operating speed
(3) ease of programming
(4) solution repeatability
(5) solution reversibility
(6) modularity
(7) expandability
(8) adaptability

Their import in configuring the DIC is discussed in the following sections.

B. Accuracy and Solution Speed

With incremental computation, the largest errors encountered are the result of quantization and truncation. Typically, the worst possible error should not exceed 2^{-n} if n is the length of a whole word in number of bits (Mayorov, 1964). In rectangular integration, the error ϵ for a monotonic continuous curve is given by $\epsilon < (y_n - y_o)\ \Delta X$

(Braun, 1963). The precision of the computation varies directly with the length of the word and, unlike general-purpose computers with bit-parallel word serial processing, the solution speed is inversely proportional to the word length and therefore to precision. Basically, precision can be increased without limit by using longer words at the expense of more time, or solution speed can be increased by sacrificing precision. Both alternatives are attractive and their advantages can be selectively exploited, depending on the application.

As noted in Chapter II, the accuracy of calculations can be improved by using higher order integrating algorithms such as extrapolating trapezoidal integration. Another method is to employ multi-bit increment transfers to reduce the errors introduced by truncation of the integral increment. Nilsen (1968) has shown that multi-bit increment transfers permit the use of shorter word lengths and resultant savings in solution time without the normally associated penalty of loss of accuracy. His method, however, does introduce the restriction that integration can be accomplished only with respect to time. Other problems associated with multi-bit transfer are discussed in Chapter IV.

In addition, accuracy can be improved by the use of floating-point arithmetic. Although this improvement is less than that achieved by multi-bit transfer, floating-point arithmetic does have the additional benefit of considerably reducing and possibly eliminating the often very difficult problem of scaling.

High operating speed can be achieved by total parallel processing; the result, however, would be incompatible with requirement 3. Serial-processing machines operate at high solution speeds for simple equations (equations whose solution requires only a few integrations per iteration) but, as the complexity of the equation increases, the solution time increases proportionately without bound. One solution to this dilemma is a machine that employs serial-parallel processing.

C. Ease of Programming

The usefulness of any device is directly related to the ease of use by the programmer. By incorporating provisions that allow for automatic

editing and programming wherever possible, the proposed system can be employed without any extra effort when attached to a GPC; in fact, given a DIC/GPC system with a built-in translator program and given a problem that contains differential equations, it would require considerably more effort on the part of the programmer to avoid using the DIC.

The automatic-editing capability is one of the key features necessary for successful operation of the system in the computer-center environment.

D. Solution Repeatability

Repeatability of operations or calculations is necessary to ensure accuracy; furthermore, it allows for some fault detection. Because the operating parameters and the initial conditions stored in the DIC are not modified or destroyed during processing, it is always possible to stop at any point and repeat the solution from its initial value.

In control-system applications, a given equation often must be solved repetitively with only a few changes in parameters, and only these initial conditions or parameters must be entered while all others are retained. A similar situation occurs when searching for the solution of problems with given initial and terminal boundaries, where some initial conditions must be changed until the proper solution is found.

E. Solution Reversibility

The DIC is capable of reversing the direction of computation; therefore, we can stop the solution at some point, reverse, and retrace it to its initial value. Given the function value at some point in time t_j, we can compute the solution by using a negative time derivative and find the solution for the interval from t_i to t_j where $i < j$.

It should be noted, however, that not all solutions are reversible. The conditions of reversibility for the solution of linear difference equations with constant coefficients are that the highest and lowest ordered difference terms of the functions must have coefficients of unity (Monroe, 1962).

F. Modularity and Expandability

The size of the DIC sets a limit on the complexity of problems that can be solved. This complexity varies with the order, degree, and the number of equations. The question then is how small the machine can be without severely restricting its usefulness. In addition, to retain high speeds, we wanted to avoid continuously increasing solution time with increasing complexity of the equations to be solved. The answer proved to be a modular system with serial-parallel processing. Each small module is large enough only to solve a reasonable range of problems; for more complex problems, it is only necessary to add additional modules. The required connection between modules is minimal and, if not used by another module, each module can operate independently on different problems. The result is a modular system that can be closely matched to the needs of the user.

G. Adaptability

It is important that the system be adaptable. The design is such that with a proper I/O buffer the DIC can be connected to almost every existing GPC because the actual operation of the DIC is independent from the GPC; the general-purpose computer is used only to translate the equation to be solved into a machine program and as an I/O device for the DIC. In this configuration, the length of time required for the programming and execution of problems containing differential equations can be reduced considerably. The efficiency of the total computing system is greatly increased because the DIC can solve the differential equations much faster than the GPC and, as a result, the GPC is free to execute other portions of the program simultaneously. As noted above, the DIC can operate totally independently, which is particularly useful in control systems and circuit applications. The DIC can realize filters, extract Fourier coefficients from some signal, or monitor and control processes (Yu, 1967; Raimondi, 1971). In these applications, the program is usually used repetitively and can be entered or changed manually.

Chapter IV

NUMBER REPRESENTATION

A. Binary Number System

Of the many possible number systems, the binary number system using 2's complement arithmetic is the most logical choice not only for the DIC operation but to ensure compatability with other computing systems. Because the basic-unity increments represent the smallest possible change of a word (a binary number), the value of the increment is limited to 0, +1, or -1. If a single bit is used to represent these increments, then a "1" represents +1, a "0" represents -1, and an alternating string of "1" and "0" represents 0. Generally, this method is called "binary communication of increments" and was introduced with the design of the MADDIDA.

To avoid the problem of "zero oscillations," ternary representation of increments can be employed. Here we use two bits, usually one sign-bit and one magnitude-bit, allowing the representation of the three desired states (0, +1, -1) and leaving one unused state (-0).

B. Circular Number System

Binary numbers and 2's complement arithmetic leads to a circular number system (Braun, 1963; Mayorov, 1964). If we start with some number and continuously add positive increments, it will eventually reach the positive maximum; with the next positive increment, the number will go to the minimum value. The reverse process occurs if we have negative increments. For example, if the range of a number is -N to N-1, then when increasing we would have 0, 1, 2, ..., (N-1), -N, -(N-1), ..., -2, -1, 0, ...; decreasing, we find the same series but in the reverse order.

This can be illustrated by considering a simple example of a binary number register restricted to three bits. Starting from zero, we add a single bit at a time to obtain a series of eight states, as shown in Table 3. The respective decimal values are tabulated in the third column. If we consider the highest order bit to be the sign-bit of a 2's complement representation, then the decimal values of the binary numbers appear in the fourth column and we obtain (0 1 1) + (0 0 1) = (1 0 0) which,

Table 3

BINARY REPRESENTATION OF NUMBERS

State	Binary Numbers	Decimal Value	Decimal Value for 2's Complement Representation
a	0 0 0	0	0
b	0 0 1	1	1
c	0 1 0	2	2
d	0 1 1	3	3
e	1 0 0	4	-4
f	1 0 1	5	-3
g	1 1 0	6	-2
h	1 1 1	7	-1

in decimals, is (+3) + (+1) = (-4). A state diagram for this table would show eight states connected in a ring such that there is a path from every state to both of its nearest neighbors. One can see that using 2's complement representation and allowing overflows will result in a circular number system.

Figure 7 is a graphical representation of the circular number system. Increasing numbers move counter-clockwise on the circle; decreasing (positive or negative) numbers move clockwise. An overflow occurs whenever point S is crossed in the counter-clockwise direction and the value of the number goes from M-1 to -M; an underflow occurs whenever S is crossed in the clockwise direction and the number value goes from -M to M-1. The negative portion of the circle is larger by one unit increment because, for some n-bit number, M-1 corresponds to $2^n - 1$ and -M corresponds to -2^n.

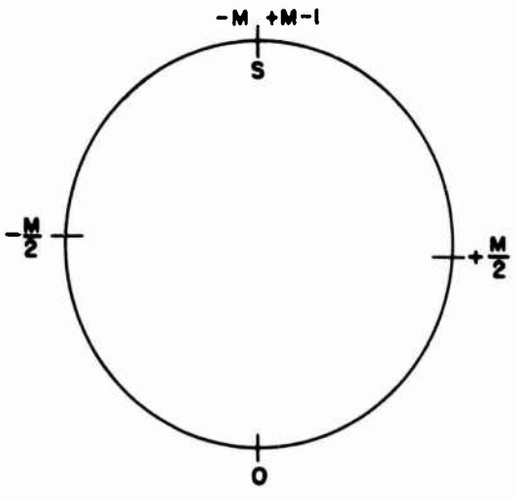


Fig. 7. CIRCULAR NUMBER SYSTEM.

This circular number system produces generally accurate results if taken over a long period of time. It can result in masked instabilities, however, if the increment of a number is zero averaged over a period

of time (but not instantaneously) and if the value of the number is at or near either the positive or negative limit.

C. Two-Loop Number System

The remainder of this chapter deals with a solution to this difficulty. Let the states in Table 3 be rotated such that the column begins with state e and ends with state d. We break the ring by not allowing any transition to go from e to d or from d to e; instead, let the (+1) transition from d go to a and the (-1) transition from e go to h. The result represents a two-loop number system, with the two loops joined by transitions between states a and h.

If the range of a number is -N to N-1, for example, and if the number continuously increases starting with some negative value -k, we obtain the series -k, (-k + 1), ..., -1, 0, +1, +2, ..., (N-1), 0, +1, +2, ..., and so on. A continuously decreasing number starting with some positive value k results in a similar series: +k, k-1, ..., +2, +1, 0, -1, -2, ..., (-N + 1), (-N), -1, -2... .

Using binary numbers with 2's complement representation, we have one more negative state than positive states for any given number of bits. If the word length is n bits (plus sign-bit), therefore, the maximum value is $(2^n - 1)$ and the minimum value (negative) is -2^n. The loop interval, however, must be the same for the positive and negative loops. As shown in Fig. 8, the return in the positive loop is to zero and the return in the negative loop is to -1. This results in a one-unit increment separation of the two loops but ensures equal loop intervals. It is also possible to consider the 2's complement representation of (-2^n) to represent, instead, the negative equivalent of

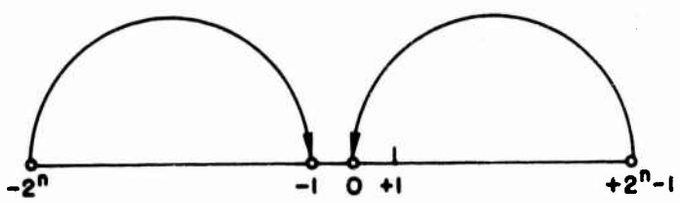


Fig. 8. TWO-LOOP NUMBER SYSTEM.

zero (-0); in the negative loop, we would have $(2^n - 1)$ steps going from -0 to $-(2^n - 1)$ instead of going from -1 to -2^n.

This two-loop number system eliminates any instabilities or oscillations such as those that occur in the circular number system because the return after an overflow or underflow is to a value other than the minimum or maximum. Table 4 compares the behavior of the two systems for a series of increments of the dependent variable which in two places contains an "average zero derivative." Figures 9 and 10 illustrate the staircase approximations of this function, emphasizing the difference between the circular and two-loop number systems. The respective $f(n)_i = \sum_{j=1}^{i} \Delta Z_j$ are plotted. As can be seen from both the table and Fig. 10, the "zero oscillations" of the circular number system are not predictable.

Since the weight of the increment ΔZ is determined by the loop length, it is clear that, given the same number of bits for both systems, the weight of ΔZ in the two-loop system will be 1/2 the weight in the circular number system and ΔZ will occur on the average at twice the rate of that in the circular system.

D. Overlapping Loops

The two number systems described can be considered to be two extremes. An interesting variation occurs if we extend the two loops such that they overlap but are not identical, as shown in Fig. 11. As an example, let the positive loop return to -3 and the negative loop return to +2. Continuously positive increments would result in the following series:

-n, -n + 1, ..., -3, -2, -1, 0, +1, +2, +3, ..., n-1, -3, -2, -1, 0, +1, ...

and negative increments would generate

n-1, n-2, ..., +3, +2, +1, 0, -1, -2, ..., -n + 1, -n, +2, +1, 0, -1, ...

The second column in Table 5 tabulates the behavior of this number system for the same series of increments used in Table 4 and can be

Table 4

GENERATION OF INCREMENTS FOR CIRCULAR AND TWO-LOOP NUMBER SYSTEMS

Arbitrary Input (Δn)	Circular Number System (n)		Two-Loop Number System (n)	
	Remainder	Increment ($=\Delta Z$)	Remainder	Increment ($=\Delta Z$)
0	0 0 0	0	0 0 0	0
+1	0 0 1	0	0 0 1	0
+2	0 1 1	0	0 1 1	0
+2	1 0 1	+1	0 0 1	+1
+2	1 1 1	0	0 1 1	0
+2	0 0 1	0	0 0 1	+1
+2	0 1 1	0	0 1 1	0
+1	1 0 0	+1	0 0 0	+1
-1	0 1 1	-1	1 1 1	0
+1	1 0 0	+1	0 0 0	0
-1	0 1 1	-1	1 1 1	0
+1	1 0 0	+1	0 0 0	0
+3	1 1 1	0	0 1 1	0
+2	0 0 1	0	0 0 1	+1
+1	0 1 0	0	0 1 0	0
-1	0 0 1	0	0 0 1	0
+1	0 1 0	0	0 1 0	0
-1	0 0 1	0	0 0 1	0
+1	0 1 0	0	0 1 0	0
+1	0 1 1	0	0 1 1	0
+1	1 0 0	+1	0 0 0	+1
+1	1 0 1	0	0 0 1	0
-2	0 1 1	-1	1 1 1	0
-2	0 0 1	0	1 0 1	0
-2	1 1 1	0	1 1 1	-1
-2	1 0 1	0	1 0 1	0
-1	1 0 0	0	1 0 0	0
-1	0 1 1	-1	1 1 1	-1
-1	0 1 0	0	1 1 0	0
+1	0 1 1	0	1 1 1	0
-1	0 1 0	0	1 1 0	0
+1	0 1 1	0	1 1 1	0
-1	0 1 0	0	1 1 0	0
-1	0 0 1	0	1 0 1	0
$f(n)_i = \sum_{j=1}^{i} \Delta Z_j$				

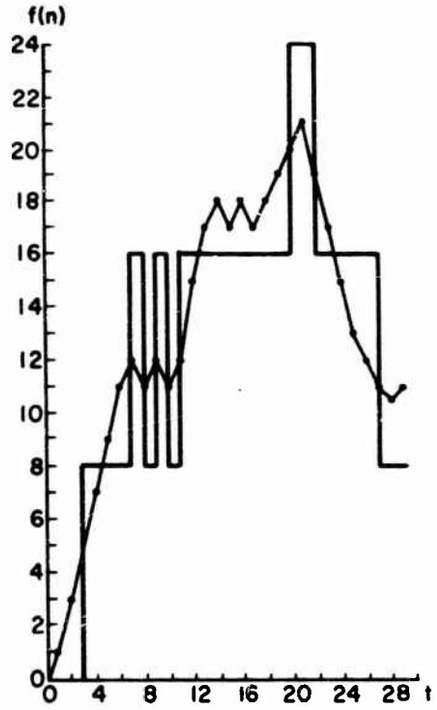


Fig. 9. INCREMENTAL INTEGRATION, USING CIRCULAR NUMBER SYSTEM.

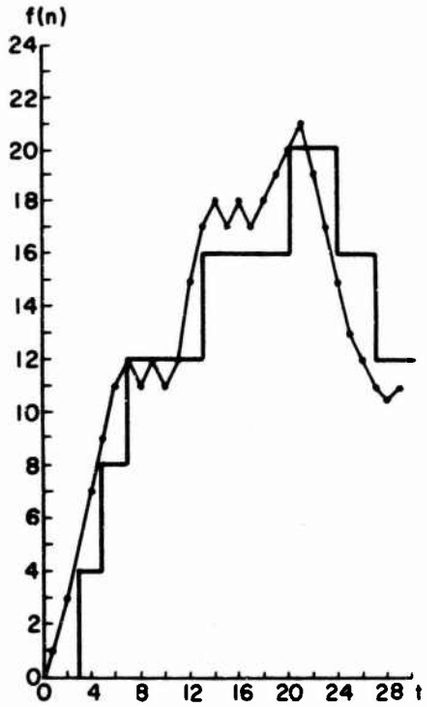


Fig. 10. INCREMENTAL INTEGRATION, USING TWO-LOOP NUMBER SYSTEM.

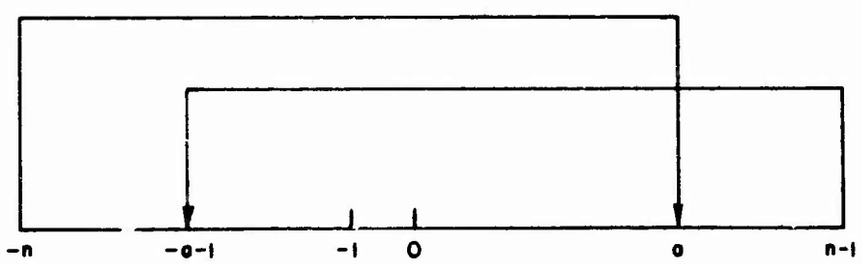


Fig. 11. TWO-LOOP NUMBER SYSTEM WITH OVERLAPPING LOOP.

Table 5

GENERATION OF INCREMENTS FOR SYSTEMS WITH OVERLAPPING LOOPS

Arbitrary Input (Δn)	1-Bit Hysteresis		2-Bit Hysteresis	
	R	ΔZ	R	ΔZ
0	0 0 0	0	0 0 0	0
+1	0 0 1		0 0 1	
+2	0 1 1		0 1 1	
+2	1 1 0	+1	1 1 1	+1
+2	0 0 0		0 0 1	
+2	0 1 0		0 1 1	
+2	1 0 1	+1	1 1 1	+1
+1	1 1 0		0 0 0	
-1	1 0 1		1 1 1	
+1	1 1 0		0 0 0	
-1	1 0 1		1 1 1	
+1	1 1 0		0 0 0	
+3	0 0 1		0 1 1	
+2	0 1 1		1 1 1	+1
+1	1 0 1	+1	0 0 0	
-1	1 0 0		1 1 1	
+1	1 0 1		0 0 0	
-1	1 0 0		1 1 1	
+1	1 0 1		0 0 0	
+1	1 1 0		0 0 1	
+1	1 1 1		0 1 0	
+1	0 0 0		0 1 1	
-2	1 1 0		0 0 1	
-2	1 0 0		1 1 1	
-2	0 1 0	-1	1 0 1	
-2	0 0 0		0 0 1	-1
-1	1 1 1		0 0 0	
-1	1 1 0		1 1 1	
-1	1 0 1		1 1 0	
	+3 0 1 1		+3 0 1 1	
	+2 0 1 0		+2 0 1 0	
	+1 0 0 1		+1 0 0 1	
	0 0 0 0		0 0 0 0	
	-1 1 1 1		-1 1 1 1	
	-2 1 1 0		-2 1 1 0	
	-3 1 0 1		-3 1 0 1	
	-4 1 0 0		-4 1 0 0	

compared to the behaviors of those systems. Figure 12 illustrates the staircase approximation for the same function, using the number system with a slight hysteresis of one-bit. "Hysteresis" here means that the loops are not separated by one or more bits and that the returns from the maximum and minimum are to two different values. The third column in Table 5 lists the same function, using a number system with a two-bit hysteresis; its staircase approximation is shown in Fig. 13. Although these systems do not eliminate all instabilities, they do prevent oscillations in the case of very small function changes at or near the maximum or minimum level. Input sequences that conceivably could cause instabilities are not likely to be encountered.

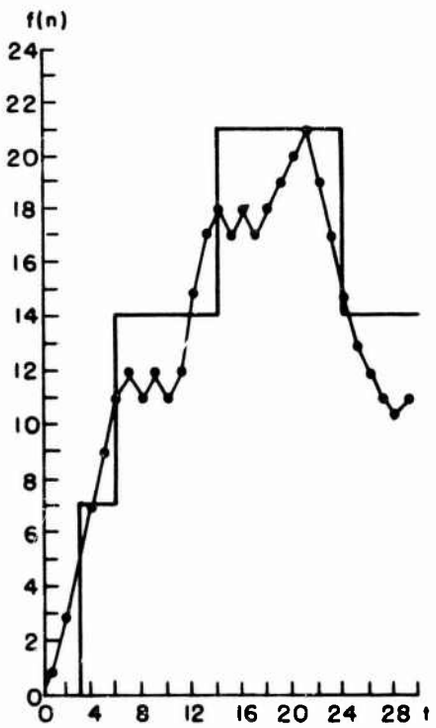


Fig. 12. INCREMENTAL INTEGRATION, USING A NUMBER SYSTEM WITH 1-BIT HYSTERESIS.

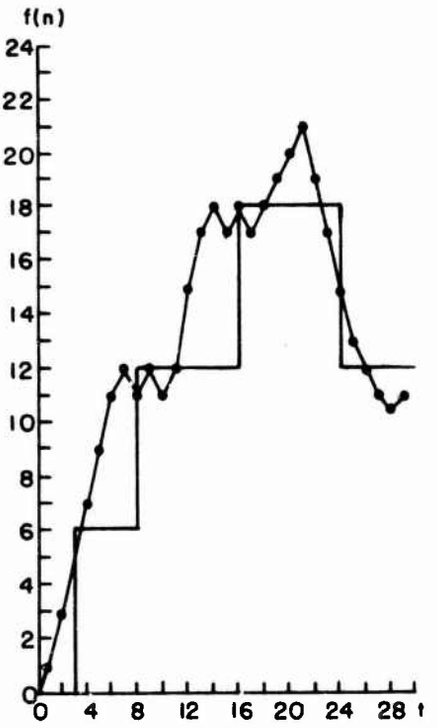


Fig. 13. INCREMENTAL INTEGRATION, USING A NUMBER SYSTEM WITH 2-BIT HYSTERESIS.

E. Logical Implementation

1. Circular Number System

The logical implementation of the circular number system requires only 2's complement arithmetic and normal overflow detection. This means that an increment is generated whenever there is a carry-bit (or borrow when subtracting) into but not out of the most significant bit (the sign-bit) or when there is a carr -bit out of but not into the most significant bit. The sign of the increment is always equal to the sign of the number before addition or subtraction. The implementation of the two-loop system varies slightly.

2. Two-Loop Number System

Let R, Y, and R^* be the sign-bits of the previous remainder, the integrand, and the new remainder, respectively, using 2's complement representation ("0" = positive, "1" = negative). Here, C is the carry into the sign-bit when adding or the borrow when subtracting; S is the add/subtract control bit and is "1" for addition and "0" for subtraction. We then want R^* and $|\Delta Z|$ as a function of R, Y, and S. The function F is necessary to eliminate C.

From the map of R^* and ΔZ (Fig. 14a), we can derive the equation

$$R^* = RY'S' + RYS + RY'C'S + R'YC'S + R'Y'CS' + RYCS' \qquad (4.1a)$$

or

$$R^* = \{[(R \oplus Y \oplus S)R]' \; [(R \oplus Y \oplus C)(R \oplus Y \oplus S)']'\}' \qquad (4.1b)$$

If F is the output of a full adder/subtracter on R, Y, C, and S, we see from the map of F (Fig. 14b) that

$$F = R^* \quad \text{if} \quad |\Delta Z| = 0 \qquad (4.2a)$$

and

$$F \neq R^* \quad \text{if} \quad |\Delta Z| = 1 \qquad (4.2b)$$

In the latter case $F = R'$.

SC \ RY	00	01	11	10
00	0,0	0,+1	0,0	1,0
01	1,0	0,0	1,0	1,-1
11	0,+1	0,0	1,0	0,0
10	0,0	1,0	1,-1	1,0

R^*, ΔZ

(a)

SC \ RY	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	1	0	1	0
10	0	1	0	1

F

(b)

SC \ RY	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	1	0	1	0
10	1	0	1	0

$R \oplus Y \oplus S$

(c)

Fig. 14. KARNAUGH MAPS FOR INTEGRAL OVERFLOW GENERATION.

The function $R \oplus Y \oplus S$ (Fig. 14c) covers all the cases where $F \neq R^*$; therefore, R^* can be expressed as

$$R^* = F(R \oplus Y \oplus S)' + (R \oplus Y \oplus S)\,R \tag{4.3}$$

Similarly from the maps,

$$|\Delta Z| = R'Y'SC + R'YS'C' + RYSC' + RY'S'C = (R \oplus Y \oplus S)(Y \oplus C) \tag{4.4}$$

but

$$Y \oplus C = F \oplus R \tag{4.5}$$

because

$$F = R \oplus Y \oplus C \tag{4.6}$$

Therefore,

$$|\Delta Z| = (R \oplus Y \oplus S)(F \oplus R) \tag{4.7}$$

Again from the maps,

$$|\Delta Z| = (R \oplus Y \oplus S)(F \oplus R) = R^* \oplus F \tag{4.8}$$

This can be checked quickly by manipulation of $R^* \oplus F$:

$$R^* \oplus F = [(R \oplus Y \oplus S)R + (R \oplus Y \oplus S)'F] \oplus F$$

$$= (R \oplus Y \oplus F)RF' + [(R \oplus Y \oplus S)R]' [(R \oplus Y \oplus S)'F]'F$$

$$= (R \oplus Y \oplus S)RF' + (R \oplus Y \oplus S)R'F \tag{4.9a}$$

$$R^* \oplus F = (R \oplus Y \oplus S)(F \oplus R) \tag{4.9b}$$

The sign of ΔZ must always be equal to the sign of R. The ΔZ generation of the two-loop system then is identical to that of the circular number system, but the sign-bit of R is inhibited from changing whenever $|\Delta Z| = 1$.

3. Multi-Bit Transfer Two-Loop Number System

The above equations were based on unity-increment transfers, which means that ΔZ can only be 0, +1, or -1. To allow the increment of ΔZ to take on values such that $(-n) \leq \Delta Z \leq (+n)$, the ΔZ generation must be changed from detecting single-bit overflows or underflows to multi-bit detection. Clearly, we could duplicate the logic described for single-bit detection and use this same logic for every ΔZ bit. For small n, this may not be too costly but, as n increases, this approach

would become uneconomical; furthermore, each detection stage introduces some additional delay that must finally propagate up to the highest order bit.

Proper multi-bit over- or underflow detection for the two-loop number system can be accomplished by using the previously described logic inserted between the most significant bit of R and its sign-bit in the same manner as in the unity-increment case. Let that portion of R which is to be read out as ΔZ be ΔZ^*, as shown in Fig. 15.

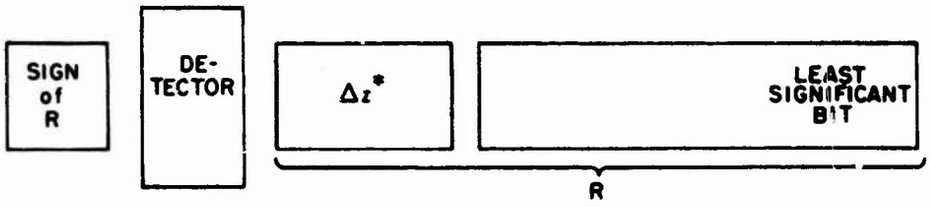


Fig. 15. INCREMENT DETECTION FOR MULTI-BIT TRANSFER TWO-LOOP SYSTEM.

All the previous equations hold, with the exception of the equations for $|\Delta Z|$, and their terms refer to the same variables as before. Again,

$$R^* = \{[(R \oplus Y \oplus S)R]' \; [(R \oplus Y \oplus C)(R \oplus Y \oplus S)']'\}' \qquad (4.10)$$

and

$$R^* \oplus F = (R \oplus Y \oplus S)(F \oplus R) \qquad (4.11)$$

however, here $R^* \oplus F$ does not give the magnitude of ΔZ but rather serves to indicate whether $|\Delta Z|$ is at its maximum. If ΔZ_m is the most significant magnitude bit of ΔZ, then

$$\Delta Z_m = R \cdot (R^* \oplus F)' \qquad (4.12a)$$

or

$$\Delta Z_m = R[(Y \oplus S) + F] \qquad (4.12b)$$

Therefore, if $R^* \oplus F = 0$, we can determine ΔZ by taking the sign-bit of R, copying this bit twice as the sign and highest order bits of ΔZ, and reading out the remaining ΔZ^* bits. All ΔZ^* bits must then be reset to be equal to the sign-bit of ΔZ. For example, let the R register contain

Sign	ΔZ^*	$R - \Delta Z^*$
0	0 1 0	1 0 1 1 0 ...

and let ΔZ^* contain three bits; then, if the least significant bit of R is on the right, the left-most bit is the sign-bit of R, the next three bits are ΔZ^*, and the remaining bits are $R - \Delta Z^*$. If $R^* \oplus F = 0$, then

$$\Delta Z = 0\ 0 \quad 0\ 1\ 0$$

and the new value for the R register is

$$0 \quad 0\ 0\ 0 \quad 1\ 0\ 1\ 1\ 0 \ldots$$

For an example with negative R, let R be

$$1 \quad 1\ 0\ 1 \quad 1\ 0\ 1\ 0\ 1 \ldots$$

If $R^* + F = 0$, then

$$\Delta Z = 1\ 1 \quad 1\ 0\ 1$$

and the new R is given by

$$R = 1 \quad 1\ 1\ 1 \quad 1\ 0\ 1\ 0\ 1 \ldots$$

If $R^* \oplus F = 1$, we have a carry (or borrow) into the sign-bit which, however, is inhibited from changing. This means that ΔZ can now be determined by again taking the sign of R as the sign of ΔZ, copying

the inverse sign of R into the most significant bit of ΔZ, and reading out the remaining ΔZ^* bits. Again, all ΔZ^* bits should be reset to be equal to the sign-bit of ΔZ.

In the case where $R^* \oplus F = 1$, however, it should be noted that all bits in the ΔZ^* register will always be equal to the sign of ΔZ and therefore need no resetting because the ΔZ^* register is always reset after readout, and the maximum value that can be added to a ΔZ^* register of n bits (excluding the sign-bit) is $(2^n - 1)$ plus a carry (or borrow) from the lower order bits of R. As a result, because ΔZ is a word that is longer by one-bit than ΔZ^*, the maximum value achieved by a ΔZ of n bits (plus sign-bit) is 2^{n-1} and the minimum value is $-2^{n-1} - 1$.

Chapter V

THE FUNCTIONAL BLOCK

Conceptually, the basic DIC module is made up of a number of identical functional blocks, each containing

(1) memory locations for the integrand, integral, and the dependent- and independent-variable increments

(2) an arithmetic unit (processor) which, when given the integrand increments and independent-variable increments as inputs, will produce the integral increments and remainder

Each block (Fig. 16) receives four inputs ΔV_i, ΔX_i, A, and B and generates as its output

$$\Delta W_i \approx A\ B\ V_i \Delta X_i \tag{5.1}$$

where V_i is the dependent variable being integrated with respect to X. The processor contains the functions of integrand-increment and integral-increment multiplications by A and B, respectively.

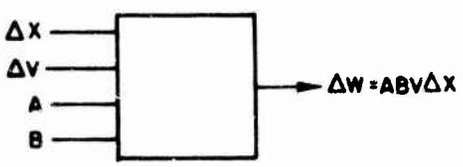


Fig. 16. FUNCTIONAL BLOCK OF PROPOSED MACHINE.

A. The Integrating Function

Figure 17 is a flow diagram of the functional block. We shall first consider the integration without multiplications by A or B.

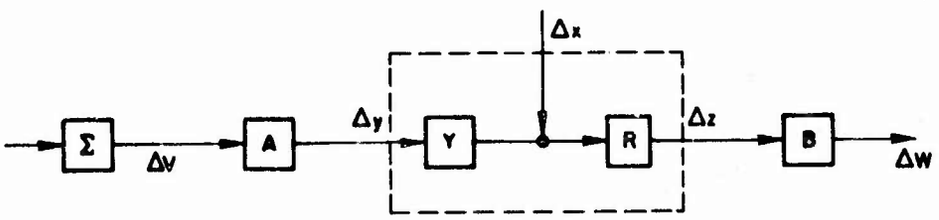


Fig. 17. STRUCTURE OF FUNCTIONAL BLOCK.

The flow diagram of this sub-block is that portion of Fig. 17 which is enclosed by the dotted line. The inputs are ΔX and ΔY and the output is $\Delta Z = Y\,\Delta X$. The equations describing the operation of the sub-block are

$$Y_i = Y_o + \sum_{j=1}^{i} \Delta Y_j \tag{5.2a}$$

$$Y_i = Y_{i-1} + \Delta Y_i \tag{5.2b}$$

and

$$R_i = R_{i-1} + Y_i \Delta X_i - M\,\Delta Z_i \tag{5.3}$$

$$\Delta Z_i = \frac{Y_i \Delta X_i + R_{i-1} - R_i}{M} \tag{5.4}$$

Explicitly,

$$\Delta Z_i = (R_{i-1} + Y_i \Delta X_i \geq M) - (R_{i-1} + Y_i \Delta X_i < -M) \tag{5.5}$$

and

$$Z_i = \left(\sum_{j=1}^{i} \Delta Z_j \right) M + R_i \tag{5.6}$$

Substituting for ΔZ_j,

$$Z_i = M\left(\frac{1}{M} \sum_{j=1}^{i} Y_j \Delta X_j + R_{j-1} - R_j \right) + R_i \tag{5.7a}$$

or

$$Z_i = \left(\sum_{j=1}^{i} Y_j \Delta X_j \right) + R_o \tag{5.7b}$$

which is approximately

$$Z_i \approx \left(\int_a^b y \, dx \right) + R_o \qquad (5.7c)$$

In these equations, M is the capacity of the integrand and remainder words and is 2^n, where n is the number of bits not including the sign-bit. Equations (5.3) to (5.5) implement the two-loop number system. Equation (5.3) can be rewritten as

$$R_i = R_{i-1} + Y_i \Delta X_i - \text{sign } (R_{i-1}) \, M \qquad (5.8)$$

since the sign of ΔZ_i is always equal to the sign of R_{i-1}, as shown in Eq. (5.5), and the magnitude of ΔZ_i is equal to 1 or 0. The effect is that if R is in the neighborhood of but less than M and the $Y_i \Delta X_i$ are positive, then R will reach (M-1) and with the next unit increment will go to zero instead of to the most negative value (-M). A similar but reverse process occurs if R and the $Y_i \Delta X_i$ are negative. In this case R will eventually reach -M and with the next increment will go to -1. Thus we have two separate loops, the positive going from 0 to M-1 and the negative going from -1 to -M.

B. Constant Multiplication

Now let us consider the total functional block. Even linear differential equations with constant coefficients require that some terms be multiplied by constants. In addition, a method of problem scaling relies on constant multiplication of the integrand and the integral increments. The functional block therefore contains both of these multiplications.

Multiplication by A will be called "pre-multiplication" since it occurs before integration; similarly, multiplication by B will be called "post-multiplication" because it is an operation on the integral result. Pre-multiplication is limited to positive constants (A) which are positive integer powers of 2 ($A = 2^a$, a = positive integer); post-multiplication is limited to positive or negative constants (B) whose

absolute value is equal to or less than 1 ($-1 \leq B \leq +1$). Using the multiplication factors A and B simultaneously allows the integral to be multiplied by any desired constant; for example, the multiplication factor of -3 is derived by setting A = 4 and B = -3/4.

The limitations set on A and B are dictated by the operating principle of the DIC. Considering post-multiplication first, it is clear that the integral increment is generated at some rate determined by both the dependent and independent variables and that this rate cannot exceed the maximum machine rate which is equal to the maximum number of iterations/sec. The highest possible rate occurs when the independent-variable increment has a rate equal to the maximum machine rate and when the absolute value of the corresponding dependent variable is at a maximum. Under these conditions, the integral-increment rate is approximately equal to the maximum machine rate. Any rate multiplication, therefore, must be limited to factors whose absolute values are equal to or less than 1.

Similar arguments apply to pre-multiplication. Given an integrand word with maximum length of n bits, the highest precision is achieved by setting the unit increment equal to $1/(2^n - 1)$ of the maximum possible integrand value. The unit increment is defined to be the smallest allowable increment of a variable or function. If in some calculation we desire maximum accuracy and if the dependent-variable increment consists of a single-unit increment, then it is not possible to multiply this increment by any factor less than 1. The restriction of the factor A to powers of 2 is a practical consideration and simplifies the logical implementation. Since the pre-multiplying factor appears mathematically outside the integral, it is not necessary to include sign reversal if that is available in post-multiplication.

If, in some calculation, precision is to be sacrificed for speed, the maximum length of the integrand words may be scaled down by uniformly pre-multiplying the integrand increments of all integrands by the same factor. This new factor then is considered to be the unity multiplication factor and any further multiplication necessary for the function is superimposed. In this case, the factor A may be less than 1 ($A = 2^a$, $a = 0, \pm 1, \pm 2 \ldots$) but still positive.

C. Implementation

The implementations of pre- and post-multiplication are quite different from each other. In pre-multiplication, the value of the incoming increment is multiplied by the multiplication factor and the result is immediately added to the integrand (pre-multiplication is the normal multiplication of two numbers). For post-multiplication, this method is only possible and necessary for multi-bit transfer machines. If we want the output to be a single magnitude bit which limits it to the values of +1, -1, or 0, then some form of rate multiplication is required. We could use a second identical functional block, set the value of the integrand to be equal to the desired post-multiplication factor, and use the output of the first block as the independent-variable increment input. The dependent-variable increment would remain zero. This, in fact, is the method normally used in DDA machines.

In a parallel-processing machine the use of integrators for constant multiplication becomes rather expensive and the availability of integrators is quickly exhausted. In a serial-processing machine this method causes the solution speed to decrease considerably, since in this case the time required for constant multiplication is equal to that required for integration.

Incorporating post-multiplication into the basic functional block somewhat reduces the extra hardware required otherwise, and also decreases the total time spent on multiplication. Both time and hardware can be saved, for example, by eliminating the circuit function which for the integration function adds the dependent-variable increment to the integrand.

Including the two multiplication functions in the functional block results in the equations below describing its operation. In these equations, M and M2 are the scale factors, R and R2 are the remainders of the integrating function and post-multiplication, respectively, and ΔV and ΔW are problem variables and are the dependent-variable increment and the multiplied integral increment.

$$A\ V_i = A\ V_o + \sum_{j=1}^{i} A\ \Delta V_j \qquad (5.9)$$

$$Y_i = A\,V_i = A\,V_{i-1} + A\,\Delta V_i \tag{5.10}$$

$$R_i = R_{i-1} + Y_i \Delta X_i - M\,\Delta Z_i \tag{5.11}$$

$$R2_i = R2_{i-1} + B\,\Delta Z_i - M2\,\Delta W_i \tag{5.12}$$

$$\Delta Z_i = \frac{A\,V_i \Delta X_i + R_{i-1} - R_i}{M} \tag{5.13}$$

$$\Delta W_i = \frac{B\,\Delta Z_i + R2_{i-1} - R2_i}{M2} \tag{5.14}$$

This equation is implemented thus:

$$\Delta W_i = (R2_{i-1} + B\,\Delta Z_i \geq M2) - (R2_{i-1} + B\,\Delta Z_i < -M2) \tag{5.15}$$

The algorithm then yields

$$W_i = M\left[\left(\sum_{j=1}^{i} \Delta W_j\right) M2 + R2_i\right] + R_i \tag{5.16}$$

Substituting for ΔW_j we have

$$W_i = M\left[M2 \sum_{j=1}^{i} \frac{B\,\Delta Z_j + R2_{j-1} - R2_j}{M2} + R2_i\right] + R_i \tag{5.17a}$$

$$W_i = M\left[\sum_{j=1}^{i} (B\,\Delta Z_j + R2_{j-1} - R2_j) + R2_i\right] + R_i \tag{5.17b}$$

or

$$W_i = M\left[\sum_{j=1}^{i} B\,\Delta Z_j + R2_o\right] + R_i \tag{5.17c}$$

Substituting for ΔZ_j results in

$$W_i = M\left[B \sum_{j=1}^{i} \frac{A\ V_j \Delta X_j + R_{j-1} - R_j}{M} + R2_o\right] + R_i \qquad (5.18a)$$

or

$$W_i = B \sum_{j=1}^{i} A\ V_j \Delta X_j + R_o - R_i + M\ R2_o + R_i \qquad (5.18b)$$

which finally can be written as

$$W_i = A\ B \sum_{j=1}^{i} V_j \Delta X_j + R_o + M\ R2_o \qquad (5.19)$$

In the preceding sections all numbers were considered to be integers. It is customary, however, to normalize all values such that the maximum absolute value of the integrand may not exceed unity. In this case, $M = 1$ and the smallest possible increment of Y is $\Delta Y = 1/2^n$, where n is again the number of bits excluding the sign-bit. It is clear, however, that the largest positive and negative values which may be contained in Y are $(1 - 1/2^n)$ and (-1), respectively. It is therefore not possible to multiply by +1 without some modification. A simple method to allow multiplication by +1 is to include an additional bit in the post-multiplier representing unity. This is not a problem in the case where separate integrators are used for constant multiplication because the second integrator in that case is deleted.

Using normalized values, the equations of the functional block are

$$Y_i = A\ V_{i-1} + A\ \Delta V_i \qquad (5.20)$$

$$R_i = R_{i-1} + Y_i \Delta X_i - \Delta Z_i \qquad (5.21)$$

$$R2_i = R2_{i-1} + B\,\Delta Z_i - \Delta W_i \tag{5.22}$$

$$Z_i = A\left[\sum_{j=1}^{i} V_j \Delta X_j\right] + R_o \tag{5.23}$$

$$W_i = A\,B\left[\sum_{j=1}^{i} V_j \Delta X_j\right] + R_o + R2_o \tag{5.24}$$

D. Post-Multiplication by a Variable

Let us again consider the basic functional block with inputs $\Delta Y, \Delta X$ and output $\Delta Z = Y\,\Delta X$ and let us assume that the equation solution requires the generation of $g\,dz \cong G\,\Delta Z = \Delta W$. Taking a separate integrator, we may obtain the increments ΔW by using ΔZ as the independent-variable increment and ΔG as the dependent-variable increment, thus giving $\Delta W = G\,\Delta Z = G\,Y\,\Delta X$. The same result may be achieved by using a built-in post-multiplier; however, the required input would be G and not ΔG which means that G must have been generated elsewhere in the problem solution. In problems requiring function multiplications, the capability of built-in post-multiplication by a function can save considerable time and hardware. A few simple examples will demonstrate potential time savings. The functional block symbol (Fig. 18) of the proposed machine has one extra output $A\,Y\,\Delta X$ which will be discussed later.

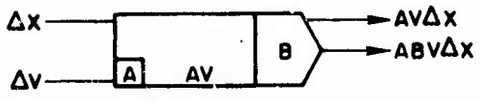
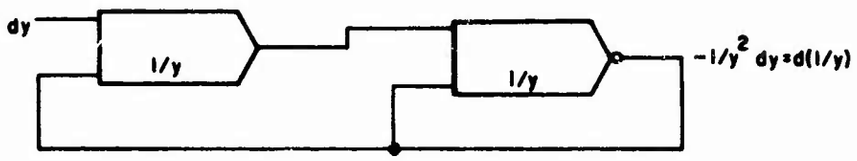


Fig. 18. FUNCTIONAL BLOCK SYMBOL.

Example 1.

Inversion: Given dy, generate 1/y. Figure 19a is the conventional DDA diagram, and Fig. 19b is the diagram using built-in post-multiplication. The inverse is generated by solving

$$d\left(\frac{1}{y}\right) = -\left(\frac{1}{y}\right)^2 dy \tag{5.25}$$



a. Conventional DDA solution

b. Solution, using built-in post-multiplication

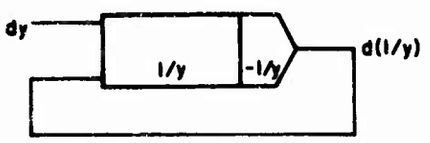


Fig. 19. GENERATION OF 1/y.

Example 2.

Generate the functions sin ωx and cos ωx by solving the differential equation

$$d^2y = -\omega^2 y(dx)^2 \tag{5.26}$$

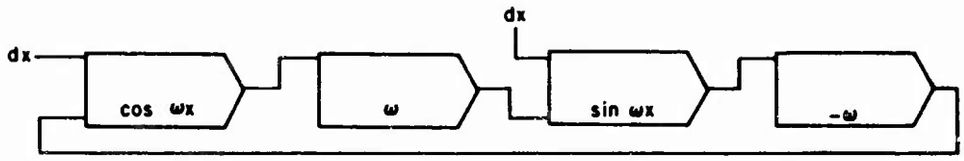
Then

$$d^2(\sin \omega x) = -\omega^2 \sin \omega x(dx)^2 \tag{5.27}$$

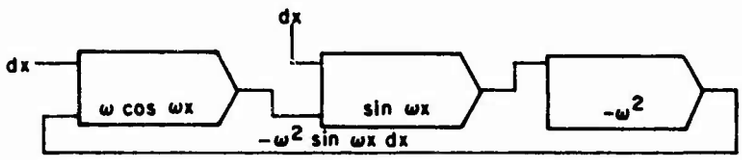
and

$$d(\sin \omega x) = \omega \cos \omega x\, dx \tag{5.28}$$

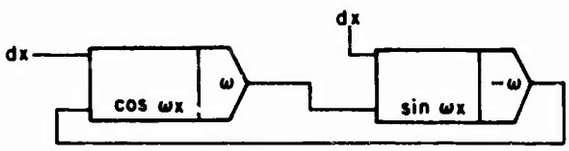
Figure 20a is the solution diagram in which both sin ωx and cos ωx are available as whole words and in incremental form. Figure 20b is an alternate diagram which uses one less integrator but generates ω cos ωx in place of cos ωx. Figure 20c is the diagram for built-in post-multiplication where both the desired functions are available and the circuit uses only two integrators.



a. Conventional DDA solution



b. An alternate solution



c. Solution, using built-in post-multiplication

Fig. 20. GENERATION OF sin ωx AND cos ωx.

Example 3.

Given dx and the constant a, generate the increments $d(x^2 + a^2)^{1/2}$ and $d(x^2 + a^2)^{-1/2}$. Figure 21a is the DDA diagram where $d\,\ell n(x^2 + a^2)$ is generated in addition to the desired outputs. Figure 21b illustrates the use of built-in multipliers. An alternate diagram (Fig. 21c) has $d\,\ell n(x^2 + a^2)$ and $d(x^2 + a^2)^{-1/2}$ as outputs. The square root is derived from

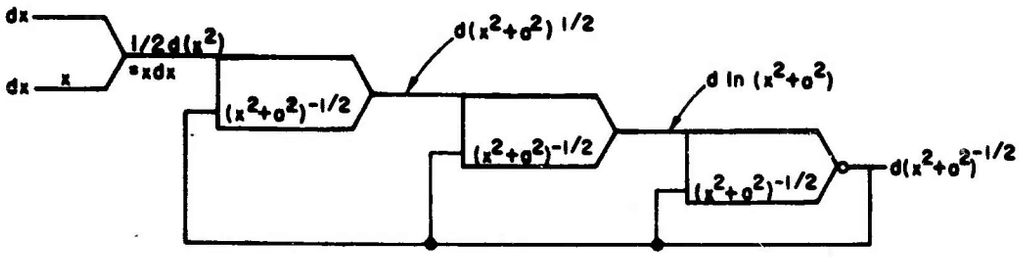
$$dy = -\frac{1}{2y}\,dx \qquad (5.29)$$

and the natural logarithm is from

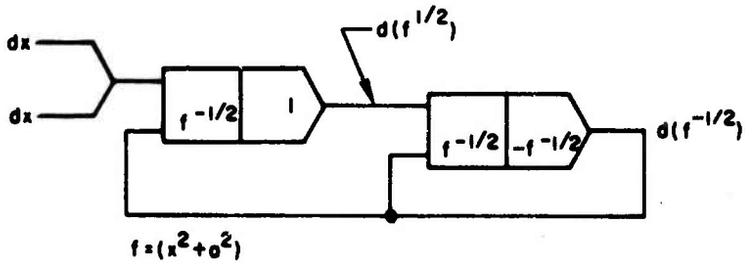
$$d\,\ell n\ y = \frac{1}{y}\,dy \tag{5.30}$$

The inverse is derived as in example 1 from

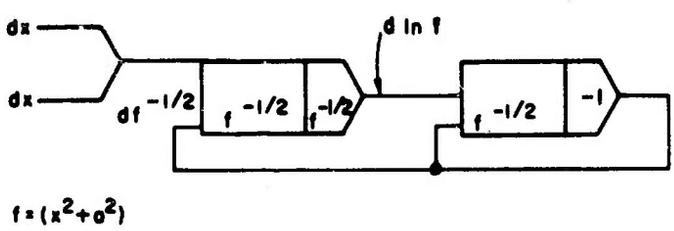
$$d\left(\frac{1}{y}\right) = -\left(\frac{1}{y}\right)^2 dy$$



a. Conventional DDA solution



b. Solution, using built-in post-multiplication



c. Alternate solution, using built-in post-multiplication

Fig. 21. GENERATION OF $d(x^2 + a^2)^{1/2}$, $d(x^2 + a^2)^{-1/2}$, AND $d\,\ell n(x^2 + a^2)$.

E. Simultaneous Integration and Post-Multiplication

As has been shown, some time saving may be gained by including post-multiplication in the functional block. The greatest advantage of built-in post-multiplication, however, is that for single-bit (plus sign-bit) communication it is possible to perform post-multiplication simultaneously with the integration. At the beginning of any integrating cycle the following values are known: ΔY_i, Y_{i-1}, R_{i-1}, ΔX_i, and the post-multiplication factor B. Given these known quantities it is possible to make some prediction about the outcome of the integrating cycle.

To proceed with the multiplication, we must know the sign and the magnitude of

$$\Delta Z_i = \frac{Y_i \Delta X_i + R_{i-1} - R_i}{M} \tag{5.31}$$

Again using normalization to unity, we know that the magnitude of ΔZ_i can only be zero or unity. The sign of ΔZ_i may be determined as follows. The absolute value of $Y_i \Delta X_i$ must be less than or equal to M; the same is true for R_i and R_{i-1}. If we assume that R_{i-1} has some positive value $M - C$ and if

$$Y_i = M - D \tag{5.32}$$

then

$$R_i - \Delta Z_i = (M - C) + (M - D) \tag{5.33}$$

for positive $Y_i \Delta X_i$, where $0 \leq C \leq M$ and $0 \leq D \leq M$. Therefore,

$$R_i - \Delta Z_i = 2M - (C + D) \geq 0 \tag{5.34}$$

which requires that $\Delta Z_i \geq 0$. For negative $Y_i \Delta X_i$, we have

$$R_i - \Delta Z_i = (M - C) - (M - D) = D - C \leq M \tag{5.35}$$

and

$$D - C \geq -M$$

which requires that $\Delta Z_i = 0$. If R_{i-1} is negative, the results are

$$\Delta Z_i \leq 0 \qquad \text{for negative } Y_i \Delta X_i \tag{5.36a}$$

$$\Delta Z_i = 0 \qquad \text{for positive } Y_i \Delta X_i \tag{5.36b}$$

Hence, the outcome of the integrating cycle can be partially predicted by the knowledge of R_{i-1} only.

We may proceed, therefore, with the process of post-multiplication simultaneously with the integrating cycle. The result of the post-multiplication is then stored and used if $|\Delta Z_i| = 1$ and is discarded otherwise. The additional time required for post-multiplication is only one gate delay and becomes essentially insignificant in comparison to the integrating cycle time. The process is valid for both multiplication by a constant as well as a variable.

If we now consider the linear differential equation

$$d\ddot{y} + ad\dot{y} + bdy + cydt = F \cos \omega t \, dt \tag{5.37}$$

we can see that four integral outputs require multiplication by constants but are also used without the multiplication as inputs to other blocks. If the output of a functional block must be multiplied by a constant or a function, that multiplication may occur within the block only if the integral output (unmultiplied) is not used. Generally, if each functional block has only one output and if each output is used as the independent-variable increment input to a second block and not used elsewhere, the two functional blocks may be combined into one provided that the dependent variable of the second functional block is already being generated in some other block or that it is a constant. If we allow the functional block to have two outputs (the integral and the post-multiplied integral) as was shown in Fig. 18, then the only restriction for simultaneous

post-multiplication is that the multiplier must be generated elsewhere or must be constant.

F. Extended Simultaneous Integration and Post-Multiplication

Theoretically, it is possible to extend the above method to allow the multiplication of the integral increment by many functions simultaneously. For example, the three integrators in Fig. 21a which contain the dependent variable $f^{-1/2} = (x^2 + a^2)^{-1/2}$ could be included in a single functional block; although all dependent variables are identical except for one sign reversal, they could be different as long as they are available or are constants.

Simultaneous post-multiplication is meaningful only in serial-processing machines. Let us consider such a machine with an extended multiplication capability which can handle p post-multipliers. Given a differential equation and its solution diagram, all integrals are first grouped into two categories.

(1) The independent-variable input to the integral is the independent variable of the total equation, or the dependent-variable input is a function not generated elsewhere.

(2) The independent-variable input is the output of some other integrator, and the dependent variable is constant or is generated by one of the integrators in the first category.

If there are n integrators in category 1 and m integrators in category 2, the solution would require $n+m$ machine cycles per iteration for a traditional serial machine. The solution would require n cycles in the machine with extended post-multiplication provided that the number of multipliers assigned to any one of the n integrators in category 1 does not exceed p.

Although the additional time required for multiplication in the extended unit is minimal, extra time is necessary for assigning and routing the additional outputs of each unit; furthermore, if the multipliers are variables, some time must be expended to fetch the variables. As pointed

out previously, usually all integrals of category 1 require one multiplication, the probability of encountering two or more function multiplications is considerably reduced, and it would be unlikely that all the variables would be generated elsewhere. Therefore, while a single simultaneous multiplication can reduce the iteration time by 1/2, it is not economical to include the extended post-multiplication (by two or more functions) capability.

G. Multi-Bit Transfer

Simultaneous multiplication as described is not possible if a system of multi-bit increment communication is used because the magnitude of the output cannot be predicted. Furthermore, it is not even possible to predict the sign of the output. Again, the known quantities at the beginning of the integral cycle are R_{i-1}, Y_{i-1}, ΔY_i and ΔX_i. Here, however, the aligned (after scaling) maximum values of R and Y are not equal. If M_Y is the maximum value of Y and M_R is the maximum value of R, then

$$M_R \leq M_Y \tag{5.38}$$

where $M_R = M_Y$ is the case of single-bit magnitude communication and can here be ignored. If

$$R_{i-1} = M_R - C \tag{5.39}$$

and

$$Y_i = M_Y - D \tag{5.40}$$

then, for positive R_{i-1} and positive $Y_i \Delta X_i$,

$$R_i - \Delta Z_i = (M_R - C) + (M_Y - D) \tag{5.41a}$$

$$R_i - \Delta Z_i = M_R + M_Y - (C + D) \geq 0 \tag{5.41b}$$

because

$$0 \leq C \leq M_R$$
$$0 \leq D \leq M_Y \tag{5.42}$$

As a result, $\Delta Z_i \geq 0$ but the magnitude of ΔZ_i is unknown. For negative $Y_i \Delta X_i$,

$$R_i - \Delta Z_i = (M_R - C) - (M_Y - D) \tag{5.43a}$$

$$R_i - \Delta Z_i = M_R - M_Y - C + D \tag{5.43b}$$

But $M_Y > M_R$; therefore,

$$M_R - M_Y < 0 \tag{5.44}$$

D may be, but is not necessarily, greater than C.

If $C \geq D$,

$$R_i - \Delta Z_i < 0 \quad \text{and} \quad \Delta Z_i \leq 0 \tag{5.45}$$

If $C < D$, nothing at all can be said about the sign of $R_i - \Delta Z_i$; it may be positive or negative. Similar results are obtained for negative R_{i-1}. It is no. possible, therefore, to have simultaneous multiplication if multi-bit communication is used.

Floating-point arithmetic combined with floating-point single-bit magnitude communication can be handled identically to fixed-point single-bit communication with respect to post-multiplication. The only other requirement is the addition of the integral output exponent to the exponent of the multiplier. Both of these exponents are known at the beginning of the integral cycle since the exponent of ΔZ must always be equal to that of the integrand Y.

H. Floating-Point Arithmetic

DDAs conventionally use fixed-point arithmetic, with all quantities scaled such that their absolute value does not exceed unity. One exception is the BFPDDA (binary floating-point digital differential analyzer) designed by J. L. Elshoff and P. T. Hulina (1970), which uses floating-point unnormalized values throughout. This requires that at least three exponents must be used for each integral: one for the integrand Y, one for the independent variable ΔX, and one for the integral increment output ΔZ and its remainder R. The advantage is that problems may be entered without normalization; the disadvantage is that several exponents must be recalculated during every iteration for each integral. This is costly in time and hardware. The following proposed alternative, while maintaining the most important features of floating-point arithmetic (such as dynamic scaling and increased computing accuracies due to reduced delays), uses a single exponent for all quantities in any one integral by employing normalized values of the independent variable.

The system is best explained by beginning with a conventional fixed-point DDA integrator. The absolute values of the integrand Y and the remainder R are less than or equal to unity. The normalized independent-variable input ΔX and the normalized output ΔZ are ± 1 or 0, and the dependent-variable input ΔY has a magnitude of less than 1. Let us now assume that R and Y are divided by 2, which is equivalent to a right shift by one bit. To maintain the balance, the ΔY inputs must be divided by 2 and the value of ΔZ must be multiplied by 2. Nothing however changes for the independent-variable input. If we use exponents of 2 to indicate the number of right shifts, then the original value of $Y = .y_1y_2y_3$ becomes $Y = .Sy_1y_2y_3 * 2^1$, where S is the inserted bit which is equal to the sign-bit of Y. Because ΔY and R are shifted simultaneously and by the same number of bits as Y, their exponents are identical to the exponent of Y and may be deleted. The output ΔZ is generated in the same way as before, and the required multiplication can be achieved by merely appending the exponent of Y. Thus only a single exponent storage and a single exponent calculation are required for the basic integrating cycle.

It was previously assumed that the incoming ΔY increment had the same exponent as Y. This is true if we begin with a completely scaled problem setup. Once any one of the integrands is shifted, however, the assumption no longer holds. Because several integral outputs may be summed to constitute a ΔY, it is necessary to equalize all of their exponents to be that of Y which, of course, is also true for the BFPDDA. If the inputs rather than the outputs are stored, only one such output will arrive at any one time; therefore, each increment of ΔY may be scaled as it arrives and the stored ΔY will always have the proper exponent. Using the stored input method does, however, create one problem. Each output can be used as the input to many other blocks and, therefore, is to be stored simultaneously in the ΔY storage of any or all functional blocks. This requires that the exponents of all ΔY (which are the exponents of all integrands) must be available at the end of each processing cycle. This implies that, to save storage space, the exponents of each block should be stored in their respective ΔY storage rather than with the integrand.

I. Floating Point Post-Multiplication

A problem arises when the floating-point output of one integral is used as the independent variable of another. If the exponent associated with the independent-variable increment is greater than zero, multiple integrating cycles are required; if the exponent is less than zero, fractional cycles are necessary. Most systems, including the machine here proposed, however, allow only one cycle per iteration.

Let us first consider floating point post-multiplication. As all other quantities in the system, the multiplier is a floating-point number such that the absolute value of the mantissa is less than unity. The integral increment, however, is a floating-point number with a mantissa of absolute value unity or zero, where the sign of the mantissa is predictable, as described for the fixed-point system. Multiplication of two floating-point numbers is achieved by multiplying the mantissas, adding the two exponents, and rescaling the result into standard

form. In our case, however, the result does not need to be rescaled because the exponent is adjusted after transmission of the result to the appropriate storage locations. Multiplication of the integral output mantissa may therefore occur simultaneously with the integrating cycle and, because both exponents are known at the beginning of the integrating cycle, post-multiplication in a floating-point system may be simultaneous.

As shown in Section D, whenever an integral output is used as the independent-variable input, the resultant function can be considered as a post-multiplication. In these cases, the integrating cycle is identical to the one described above except that the output exponent is the sum of the dependent- and independent-variable increment input exponents. No storage is necessary for the independent-variable input exponent because it is retransmitted for every iteration.

Chapter VI

THE MULTI-MODULE SYSTEM

A. Processing Methods

Assuming that we have a differential equation requiring m separate integral calculations per iteration, we can visualize the equation as m points in a two-dimensional space, each representing one integral function. To solve the equation, the m points must be interconnected according to the solution diagram.

Now, any DDA machine can be represented pictorially as a matrix of n points, each representing one integrator and each having two inputs and one output. If we assume full parallel operation, where all n points have their own processor, then, after each processing cycle, up to n signals must be routed to a maximum of $n^2 + 2n$ destination points. This is based on the assumption that each integral output can be used as the independent variable of two other integrals and as a component of the dependent-variable input of all integrals including itself. This is not a practical scheme for any method other than patchboard programming. The complete parallel machine, therefore, is not suitable under the conditions described in Chapter III.

Let us now examine the other extreme and assume that a single processor is available. This processor then operates on each integral in some predetermined order and, after each processing cycle, one signal must be routed to as many as $n + 2$ points. The required interconnection becomes simple enough to be handled automatically with a reasonable amount of hardware. As n becomes large, however, the iteration and solution times increase proportionally to n. This does not satisfy the condition of high solution speed discussed in Chapter II and, therefore, is not suitable.

Between these two extremes is the alternate method of serial-parallel processing. If we consider the matrix of n points to be divided into m smaller submatrices of ℓ points each and if we have m processors (one for each ℓ point matrix), then we have serial processing within each submatrix but we may process all submatrices in parallel with each other. The iteration time would increase as the number of required

integrals increases but could not exceed ℓ integrating cycles. Within each submatrix, one signal must be capable of being routed to as many as $(\ell + 2)$ inputs after each integrating cycle. The serial-parallel processing approach, therefore, simplifies the interconnection problem within each module by limiting the number of signals to be routed at any one time to one signal and by limiting the number of destinations to which this signal is to be routed to $(\ell + 2)$. This approach, however, does introduce the problem of interconnection between matrices.

B. Inter-Module Communication Methods

Here again the practical solution is a compromise between the two extremes of total interconnectibility, where every point of every matrix can be connected to all other points of all other matrices and where no interconnection exists between the matrices. The first becomes impractical because of the very large expense in hardware and time. The second extreme is unacceptable because it breaks the system into separate machines that cannot communicate with each other and therefore restricts the complexity of problems that can be solved to ℓ; "complexity" refers here to the number of integrals required for the equation solution, which depends jointly on the order and degree of the equation. The alternative is to introduce restrictions and limitations that will simplify the hardware requirements and still allow a reasonable and sufficient level of interconnections between matrices. Two possible basic approaches are "vertical communication" and "horizontal communication."

1. Vertical-Communication Approach

For vertical communication, all matrices are stacked such that, if all points in each matrix are labeled 1 through ℓ, each point i $(1 \leq i \leq \ell)$ in every matrix is directly below and above the points i in the vertically adjacent matrices. Assuming total communication within each plane, we now allow each point i to communicate with the points i directly above and below. This communication scheme in itself is too restrictive to be useful; however, by using communication channeling, any point can communicate with any other point in any plane. "Communication

channeling" here means the duplication at some point in the plane of a function being generated at another point such that the output can be communicated to the required destination point.

Figure 22 is an example of communication channeling in the vertical-communication scheme. For point A to communicate with point B, we require the duplication of A at A2. It is important to note that this communication channel is unidirectional. Vertical communication may be advantageous for some problems (such as weather-prediction calculations) but, in most cases, will lead to an excessive amount of function duplications and hardware.

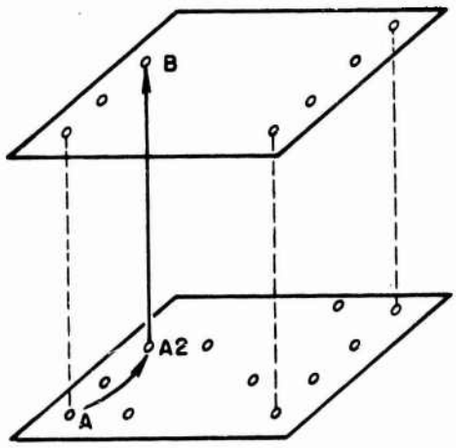


Fig. 22. CHANNELING IN VERTICAL COMMUNICATION.

2. Horizontal-Communication Apprcach

For horizontal communication, all matrices are arranged in a row. Each matrix contains two sets of points (set A and set B) such that each point in set A (of matrix n) can communicate with every point in set B contained in the matrix (n - 1), and all points in set B of matrix n can communicate with every point in set A of matrix (n + 1), as indicated in Fig. 23. The necessary hardware requirements for this inter-matrix communication scheme depends directly on the number of points contained in sets A and B. As the size of the sets A and B is reduced, however, there will be points within each matrix which are not contained in either set and, therefore, cannot communicate directly with points in other matrices. Communication channeling

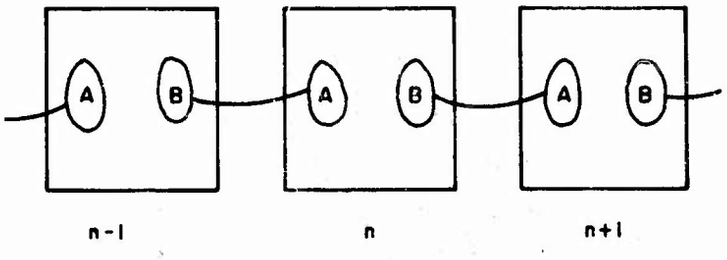


Fig. 23. HORIZONTAL COMMUNICATION.

is required in any case to allow all points to communicate with each other unless sets A and B in each matrix are identical and all points within the matrices are contained in the sets. Figure 24 is an example of the use of channeling to permit communication between points C and D in two adjacent matrices. Again there is a duplication at point C2 of the function generated at C.

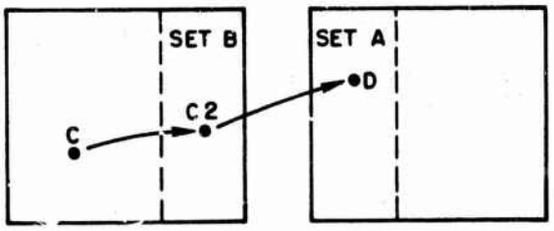


Fig. 24. CHANNELING IN HORIZONTAL COMMUNICATION.

Chapter VII

THE MODULE

A. Processing Methods

The proposed machine will be a modular system. Each module is to be complete and capable of operating independently or in conjunction with other modules. Each must contain all the necessary memory for program and data storage for some fixed number of different integrals, as well as the required arithmetic capability. The module must be automatically programmable, and it must be capable of repetitive operation and solution reversal (if the function is reversible).

The above conditions influence the selection of the construction parameters. The two most important of these are the method of processing (serial or parallel) and the method of intra-module increment communication.

The methods of parallel and serial processing have been described in the previous chapter in the context of interconnectibility. It was shown to be advantageous to use parallel processing between the modules but to use serial processing within each module. The principal disadvantage of serial processing is that the iteration time increases as the complexity of the problem increases; the principal disadvantage of parallel processing is that the difficulty of integrator interconnection prevents any practical automatic programming. The primary advantages of serial processing within the module are economy of hardware and interconnectibility. The serial-processing module requires only a single arithmetic processor. The number of integrals that can be solved within the module is limited only by the availability of memory space for program and data storage. Because only one integral is processed at a time, a single set of inputs and outputs must be routed between the integrating cycles. This, in contrast to parallel processing, allows for completely automatic programming.

B. Intra-Module Communication Methods

The DDA program is a digital model of the system under study. With few exceptions, therefore, the programs are closed-loop systems with all functions generated internally, and all inputs to the integrators are derived from the outputs of other integrators. The exceptions, of course, are those instances when the machine is used for real-time applications; in these cases, one or more externally generated functions are entered as integrator inputs. To communicate the various outputs to the desired inputs, it is necessary to employ at least temporary storage for the signals. We may choose to store the outputs and then, at the beginning of each integrating cycle, to fetch the various required outputs which may combine to form the dependent-variable input and, unless machine time is used, to fetch the output that is used as the independent-variable input. Alternately, we may choose to transmit each output as it becomes available and continuously update and store all the input functions. These two methods are called "output storage" and "input storage," respectively.

1. Function Output Storage

Function output storage requires a minimum of memory. Each functional block has two inputs; one is the dependent-variable input and is generated by combining several outputs and therefore requires full-word storage in contrast to the single-bit plus sign-bit that is necessary for the storage of the output. At the beginning of each integrating cycle, assuming that all outputs are stored, one output is fetched to be used as the independent-variable input unless the machine time is used. All outputs necessary for the dependent-variable input are then fetched and the input is formed in some arithmetic unit. Although its operation is always a summing of the various outputs, the arithmetic unit must consist of a network of adders such that all outputs can be summed simultaneously, or a considerable amount of time must be allocated to generate this input. Assuming simultaneous summing, all the required outputs must be available simultaneously. This can be achieved by using a selection matrix, but again this requires a considerable amount of hardware. The program storage necessary for the selection codes is $n \cdot (n + \log_2 n)$ bits, where n is the number of functional blocks in the module.

2. Function Input Storage

Function input storage requires memory space for two inputs for each functional block; as stated above, one of these is a full word. At the beginning of the integrating cycle, both inputs are immediately available; at the conclusion of the cycle, the output is routed to the appropriate independent-variable storage locations. The output also is routed to be used to update all appropriate dependent-variable inputs. One method is to employ a separate counter for each of these inputs; the output then updates the counters by either adding or subtracting one unit increment. The contents of the counters therefore are the dependent-variable inputs and are always current. An alternative method would be to employ a single adder and to fetch and update all selected inputs sequentially. Although the first method requires considerably more hardware, it is preferred because the second method requires too much time. Using counters simplifies the selection network by allowing count-enable lines to be directly tied to the proper section of the program word. In this case, the program storage necessary for the selection codes is $n \cdot (n + 2 \log_2 n)$ bits, where n again is the number of functional blocks within the module.

Input storage, unlike output storage, allows easy expansion of the module to a larger number of functional blocks with the increase in hardware being directly proportional to the number of added blocks.

C. Memory Organization

If we assume that for each integral (and integral remainder word stored) we have one corresponding integrand word, then, because each cycle occurs in the same time interval during each iteration, it is possible to organize the data-memory block as a push-down stack. This considerably reduces the memory necessary for program storage by eliminating the address portion from each program word. This trade-off, however, may lead to duplication and multiple storage of the same identical integrand word.

A similar situation exists for the storage of post-multiplication factors, but we have one additional consideration. Post-multiplication

by a constant B, as described earlier, is unlikely to result in much duplication if the push-down stack organization is used. If, however, we maintain random-access organization for both the integrands and the post-multiplication factors, it is possible to expand the post-multiplication concept to include function multiplication.

If we assume a module size which allows the generation of m integrals, each with a different dependent and independent variable, then that module can solve a linear constant-coefficient differential equation of order m. In the case where random-access organization is maintained for all data, the coefficients may be varying functions provided that they have already been generated in the course of the equation solution. If any of the functions must be generated separately or if the order of the equation exceeds m, two or more modules can be cascaded for horizontal communication or they may be stacked for vertical communication to provide the increased capacity necessary for the solution. The number of integrals required to solve nonlinear differential equations depends on the order as well as the degree of nonlinearity, here jointly referred to as the complexity of the equation.

Chapter VIII

THE PROPOSED MACHINE

A. Operating Procedure

The system here proposed consists of one or more identical modules which are self-contained and may operate jointly on the same problem or individually on separate problems. Figure 25 is a block diagram of one module.

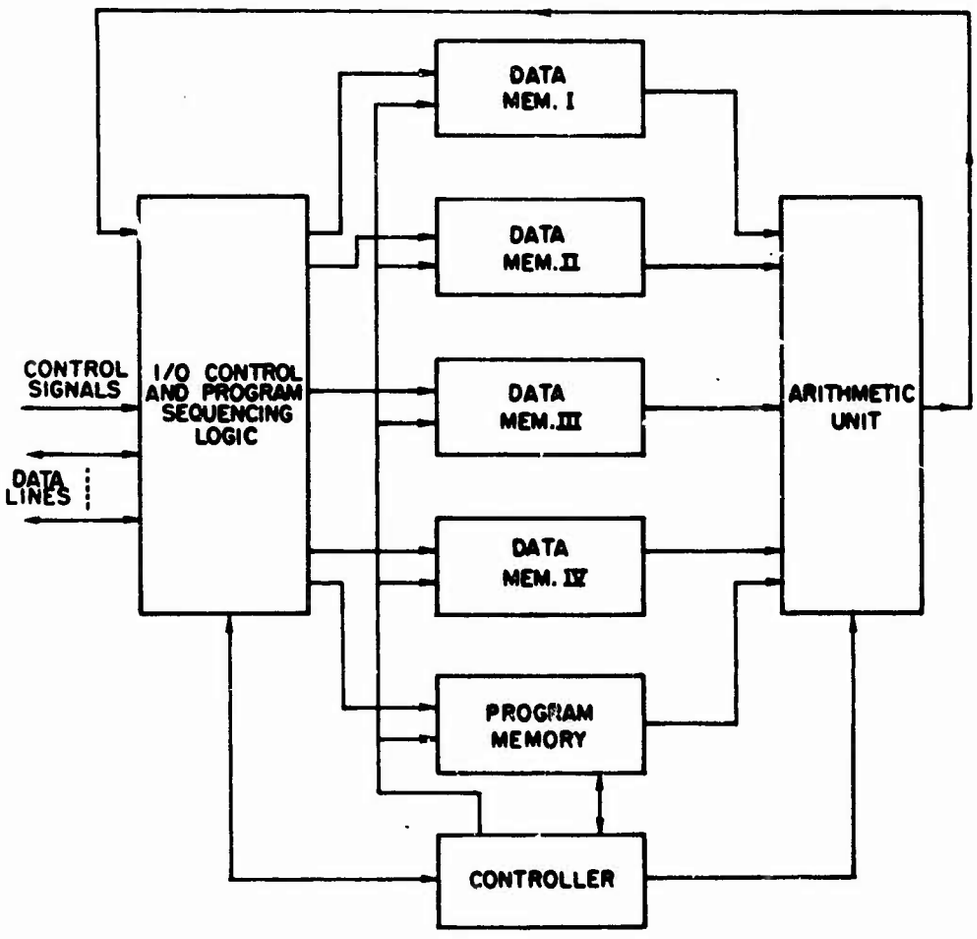


Fig. 25. BLOCK DIAGRAM OF A MODULE.

The modules follow a cyclic operating procedure. During each iteration, all integrals are processed sequentially for one integral increment. The order of processing is determined by the program and is repeated

for each iteration. The processing of a single integral increment is called a cycle and, in this system, a cycle consists essentially of the following operations:

(1) fetch the integrand increments

(2) pre-multiplication

(3) incrementation of the integrand

(4) fetch the independent variable increment

(5) generation of the integral increment and the new integral remainder

(6) post-multiplication

(7) storage of the integral increment as dependent- and independent-variable increments, as required

Figure 26 is a conceptual flow chart of the module operation. The actual time sequence of operations does not always follow this pattern because of the considerable amount of parallelism and overlapping of functions. Post-multiplication, for example, is initiated at the same time as pre-multiplication and is completed before the integral increment ΔZ is available. The result of post-multiplication then is held until the ΔZ generation is completed, whereupon we either store or discard the result depending on the magnitude of ΔZ.

B. Communication within the Module

The proposed machine will employ input variable storage. If ΔW_{ik} is the i^{th} increment output of the k^{th} integral function, then in terms of problem variables each integral increment ΔW_{ik} can be used as the dependent-variable increment $\Delta V_{j\ell}$, where $j = i + 1$ if $k \geq \ell$ and $j = i$ if $k < \ell$, for any or all integrands in the module including the case where $k = \ell$.

The increments ΔW_{ik} can also be used as the independent-variable increment $\Delta X_{j\ell}$ for any two integrals (again $j = i + 1$ if $k \geq \ell$ and $j = i$ if $k < \ell$). Each dependent-variable increment ΔV_{ik} may be a

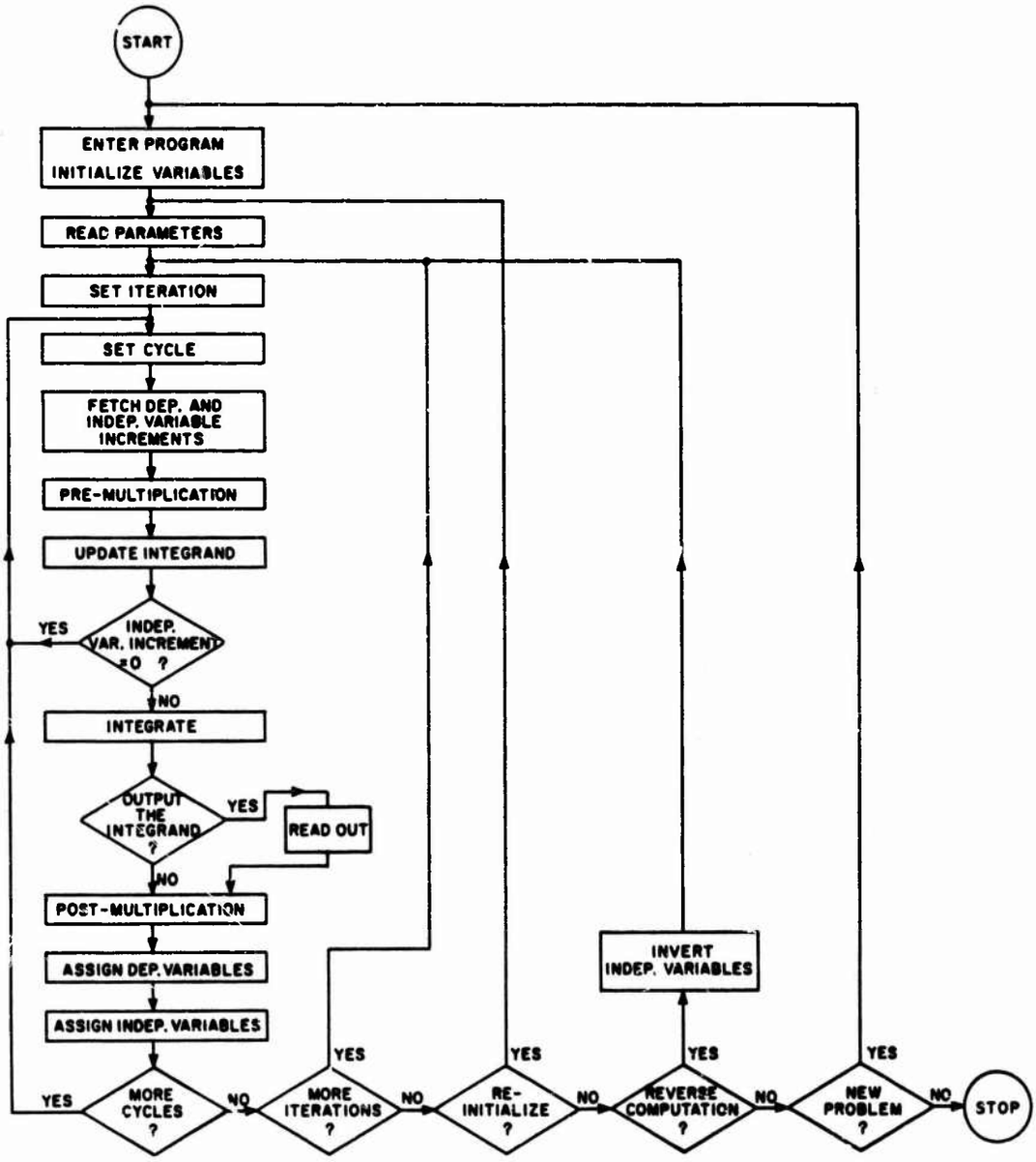


Fig. 26. FLOW CHART FOR MODULE OPERATION.

single $\Delta W_{j\ell}$ or may be formed by the summation of any or all $\Delta W_{j\ell}$. Within each module, therefore, we allow total communication for the dependent variable. The limitations on the independent variables are not as severe as may appear. If some integral increment must be used as the independent-variable increment of more than two integral functions, one simply has to generate one integral twice. In addition, the

restriction can be relaxed or even be eliminated although this would probably not be desirable. The dependent-variable increments are updated for all integrals during every cycle, and the same is true for the independent variable. Thus all inputs to any integral are ready and immediately available at the initiation of any cycle.

The memory organization is such that random access is maintained for both the integrands and post-multiplication factors. The machine, therefore, is capable of simultaneous integration and multiplication by either a constant or a variable.

C. Communication between Modules

If a multi-module system is required, inter-module communication is achieved by the "horizontal communication" method. All modules are processed in parallel.

The required hardware connections between cascaded modules consist of two sets of increment-communication links between any two modules. These links are unidirectional.

The communication between modules, while not as general and complete as within each module, allows for any or all elements of the set of the four integral increments ΔW_i $(i = m-3, m-2, m-1, m)$ of each module n to be used as the dependent-variable increments for any or all ΔV_i $(i = 1, 2, 3, 4)$ or as components thereof, of each module $n+1$. Also, any or all elements of the set of the four integral increments ΔW_i $(i = 1, 2, 3, 4)$ of each module n can be used as the dependent-variable increments for any or all ΔV_i $(i = m-3, m-2, m-1, m)$ or as components thereof, of each module $n-1$.

A reasonable and convenient size for m is 16 which means that a single module is capable of solving equations up to the 16th order, provided that the coefficients are already generated in the problem solution or that they are constants. For many applications this should be more than sufficient. If a greater capability is desired, one has only to cascade additional modules. In Fig. 27, each integral is represented as a point in a 4×4 matrix and each square plane of 16 points represents a module. Any point enclosed by a solid line has total communication with all other points within that enclosure (within the module),

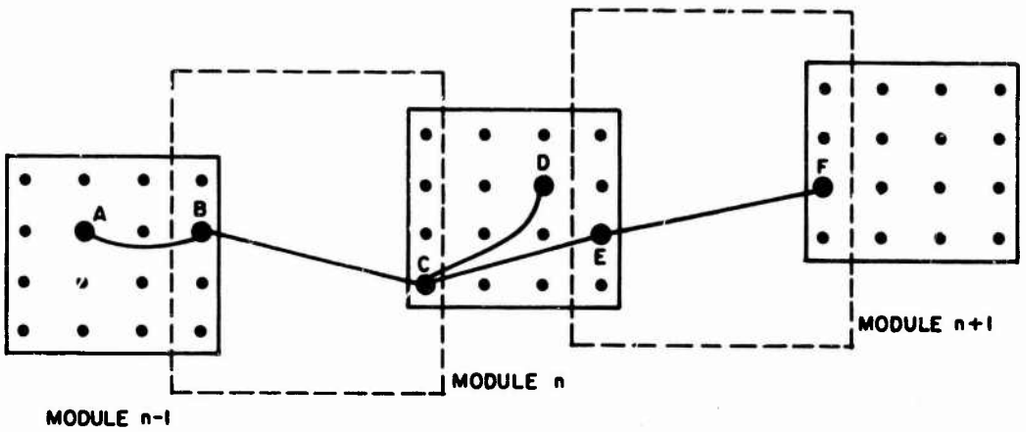


Fig. 27. MATRIX SHOWING COMMUNICATION BETWEEN MODULES.

and any point enclosed by a broken line has total communication with all other points within the broken-line enclosure (external to the module).

Each enclosure then is a domain of total communication for each point within it, and each point belongs to at least one but not more than two domains. Two points which are not contained in the same domain cannot communicate directly. Indirect communication is achieved by channeling. (For example, points A and D in Fig. 27 communicate via points B and C, and points A and F communicate via points B, C, and E.) Other communication schemes are possible and some modifications of the above are being considered; however, any method must by necessity use some kind of channeling to keep the basic organization simple.

If a particular integral output is to be used in an adjacent module, the output increment is transmitted to that module immediately after generation and is held there temporarily until the next integrating cycle has been initiated. During each integrating cycle, a time slot is reserved to store the external inputs received during the previous cycle. Because only one input can arrive during any one cycle, the routing of that signal is simplified considerably; furthermore, the interleaving of internal and external signal storage enables us to store the address of the signal in the destination module rather than in the source module.

This saves both hardware and time because the address does not need to be transmitted along with the signal between modules.

D. External Function Input

To use the machine for real-time control, it is necessary to provide for real-time function inputs. The communication links available to cascade modules to each other, if not required to connect to another module, can be used to enter external functions. A module therefore is capable of accepting up to eight external function inputs. As modules are cascaded into larger systems, only the modules at either end of the string can accept four external inputs each in this way. These external inputs also are circuited to be used for specific integrals (cycle numbers) within the modules. Additionally required external functions must use separate links going to those integrals only, which in Fig. 27 are represented as points belonging to a single domain.

E. Iteration Time

Generally, as the complexity or the order of the equations increases, the solution time increases proportionally, which is true in most numerical-solution methods. The solution time per iteration required for the DIC increases linearly with increasing problem complexity until it equals 16 cycle times (for $m = 16$); thereafter, the iteration time stays constant as additional integrations are performed simultaneously in adjacent modules. Thus the solution time per iteration for the DIC cannot exceed the time required to process 16 integrations, regardless of the problem size.

Although once established, the order of processing remains the same for every iteration; that order can be selected during program setup to allow easy communication to other modules without much function duplication or to reduce the iteration time by distributing one problem over two or more modules. In examples 1 and 2 in Chapter II, a total of six integral functions are required each if post-multiplication is not used; therefore, the iteration time is equal to six cycle times. We could, however,

distribute the integral functions over the two modules n and n + 1 such that each module contains three functions. The iteration time in this case would be equal to three cycle times or one-half of the time required previously.

F. The Processor

The processor of the proposed machine is described by the equations in Chapter V, which were written for rectangular integration. However, the processor contains one additional element which allows a choice of several integrating algorithms. Equation (5.21),

$$R_i = R_{i-1} + Y_i \Delta X_i - \Delta Z_i$$

determines the algorithm. This equation can be rewritten in more general form as

$$R_i = R_{i-1} + \left[Y_{i-1} + (K + 1)\, \Delta Y_i \right] \Delta X_i - \Delta Z_i \qquad (8.1a)$$

$$R_i = R_{i-1} + Y_i \Delta X_i + K\, \Delta Y_i \Delta X_i - \Delta Z_i \qquad (8.1b)$$

where K is a constant which determines the algorithm being used.

Four simple and very easy-to-implement algorithms are obtained by selecting K to be 0, +1/2, -1/2, or -1. If K = 0, we have the previous case of rectangular integration. Setting K = +1/2 results in the modified trapezoidal integration rule which is an extrapolating algorithm. In this case,

$$R_i = R_{i-1} + Y_i \Delta X_i + \frac{1}{2} \Delta Y_i \Delta X_i - \Delta Z_i \qquad (8.2a)$$

or

$$R_i = R_{i-1} + \left(Y_{i-1} + \frac{3}{2} \Delta Y_i \right) \Delta X_i - \Delta Z_i \qquad (8.2b)$$

The interpolating trapezoidal algorithm requires that $K = -1/2$. Then,

$$R_i = R_{i-1} + Y_i \Delta X_i - \frac{1}{2} \Delta Y_i \Delta X_i - \Delta Z_i \tag{8.3}$$

The fourth algorithm allows the proper multiplication of two variables if K is set to -1 in one of the cycles and to 0 for the other. Summing the two outputs and ignoring the remainders result in

$$\Delta(X_i Y_i) = Y_{i-1} \Delta X_i + X_i \Delta Y_i \tag{8.4a}$$

or

$$\Delta(X_i Y_i) = Y_i \Delta X_i + X_{i-1} \Delta Y_i \tag{8.4b}$$

Both of these equations represent the exact product. It should be noted that the exact product also can be achieved by using the interpolating trapezoidal algorithm ($K = -1/2$) for both functions.

To generate the new remainder R and the output ΔZ, we must first update the integrand Y. This time slot can be used to perform the addition of $K \Delta Y_i \Delta X_i$ to the old remainder. Each integrating cycle then has the following three phases.

Phase 1: Pre-multiplication

$$\Delta Y_i = A \Delta V_i \tag{8.5}$$

Phase 2: Update the integrand and modify the remainder, depending on the choice of algorithm

$$Y_i = Y_{i-1} + \Delta Y_i \tag{8.6}$$

$$R_i^* = R_{i-1} + K \Delta Y_i \Delta X_i \tag{8.7}$$

Phase 3: Generate the integral output and new remainder, and simultaneous post-multiplication

$$R_i = R_i^* + Y_i \Delta X_i - \Delta Z_i \tag{8.8}$$

$$R2_i = R2_{i-1} + B\,\Delta Z_i - \Delta W_i \tag{8.9}$$

During the processing, only phase 1 and Eq. (8.6) depend on the value of ΔX; therefore, whenever $\Delta X = 0$, Eq. (8.7) and all of phase 3 do not effect the outcome of the calculation and therefore may be deleted. Normally, this case occurs quite frequently unless ΔX is supplied by the machine time. Hence, to increase the solution speed further, the machine will operate semi-asynchronously.

Although all operations occur in the same time slot during each cycle, execution of Eq. (8.7) during phase 2 is prevented and the cycle is terminated after this phase if $\Delta X = 0$, provided that the equation to be solved is contained in a single module and that no real-time external signals are used. In real-time problems and if the problem is distributed over several modules, the processing operation must be synchronous.

G. Programming and Interface

Since the system is completely modular, the programming is identical for each module; therefore, only one module will be considered here. The module may be programmed automatically from a general-purpose computer containing the appropriate software-package or it may be programmed manually. In either case, all programming steps are identical and must be entered in the same order.

For the simplified machine which allows simultaneous post-multiplication by a constant but not by a variable (this eliminates the indirect-addressing step prior to post-multiplication), the program contains the following information:

(1) number of integral cycles

(2) integral address (cycle no.)

(3) initial condition of the integer (Y)

(4) initial condition of first remainder (R)

(5) pre-multiplication factor (A)

(6) post-multiplication factor (+/-,PY)

(7) initial condition of second remainder (PR)

(8) positive or negative independent-variable input, either machine time or transmitted signal (+/-,T)

(9) addresses of dependent-variable inputs to which output is to go (internal to module) (DYA)

(10) addresses of independent-variable inputs to which output is to go (internal) (DXA)

(11) is integral increment to be used for adjacent module (I/E)

(12) is integral increment used as output (O)

(13) addresses of dependent variable to which external input is to go (DYAE)

Steps 2 through 13 are repeated for each integral cycle. Entering the options of solution range (number of iteration cycles), repetitive operation, or solution reversal completes the program.

All this information is entered in binary form and is automatically cycled to the appropriate memory-storage locations. As an example, let us consider Eq. (2.15) in Chapter II:

$$y\ddot{y} + \dot{y}^2 + 1 = 0$$

or

$$d\dot{y} = -\dot{y}\,\frac{dy}{y} - \frac{dx}{y} \qquad (8.10)$$

The solution diagram is repeated for convenience in Fig. 28. Assuming the module is operating in conjunction with a general-purpose computer containing the translator, the following input statements are required to generate the DIC program:

```
INDV X
DEPV Y
DY1 = -(Y1)(DY/Y) - (DX/Y)
FIN
NUMBER OF EQUATIONS = 1
```

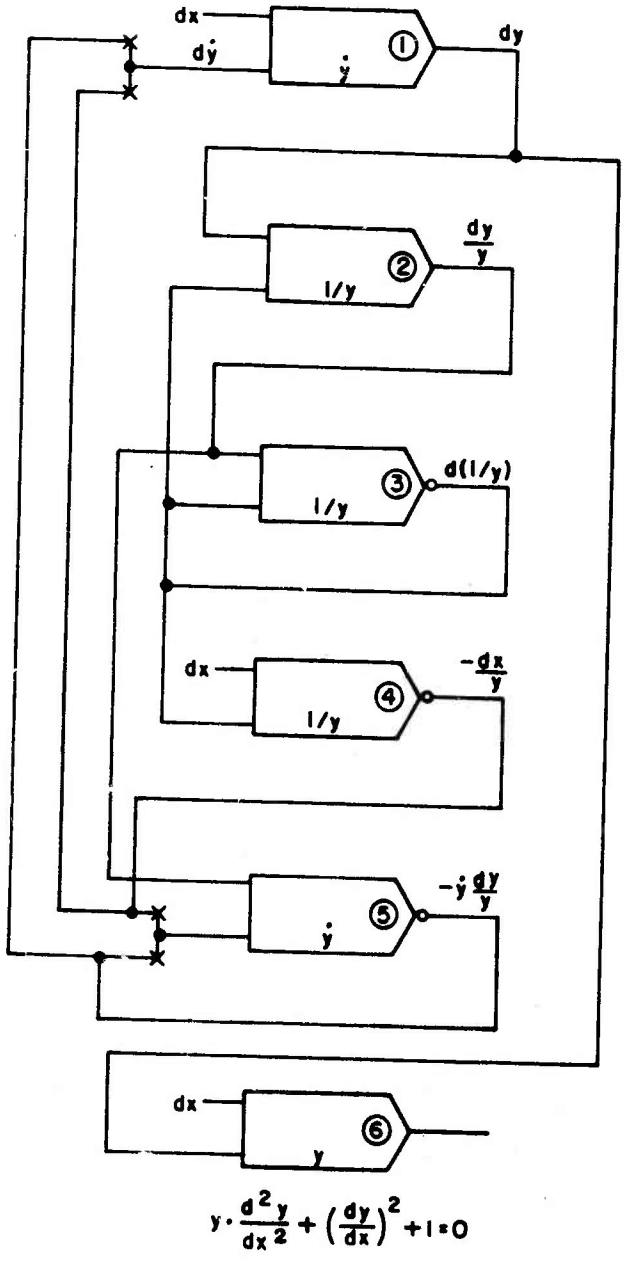


Fig. 28. SOLUTION DIAGRAM FOR $y\ddot{y} + \dot{y}^2 + 1 = 0$.

The first two statements are the declarations of the independent and dependent variables, respectively. Following the declarations is a list of all coupled equations to be solved. The form of the equation statement is very similar to that of Eq. (8.10), but the dots over the variables are replaced by a numeral to signify the order of the derivatives ($d\ddot{y}$ becomes DY2). FIN indicates the end of the equation list. If other equations (uncoupled from the previous set) are to be solved simultaneously, the FIN statement is replaced by NEXT and is followed by the new declarations. Except for the operating options, the complete DIC program for Eq. (8.10) is shown in Table 6. Again the assumption is that, at most, 16 bits are to be used and all initial conditions are normalized.

Table 6

DIC PROGRAM FOR $d\dot{y} = -\dot{y}\,\frac{dy}{y} - \frac{dx}{y}$

Number of cycles = 6, and selection codes are always (0/1).

Cycle (No.)	Y	R	A	+/-,PY	PR	+/-,T	DYA	DXA	I/E	O	DYAE
1	1.00	0.50	4	0,1.00	0.50	0,1	6	2	0	0	--
2	1.00	0.50	4	0,1.00	0.50	0,0	--	3,5	0	0	--
3	1.00	0.50	4	1,1.00	0.50	0,0	2,3,4	--	0	0	--
4	0.25	0.50	1	1,1.00	0.50	0,1	1,5	--	0	0	--
5	1.00	0.50	4	1,1.00	0.50	0,0	1,5	--	0	0	--
6	0.25	0.50	1	0,1.00	0.50	0,1	--	--	0	1	--

The combined GPC-DIC system requires an interface to channel communications of input, output, and commands, and to match the data rates of the two machines. To the GPC the interface will appear as an I/O channel and the DIC as some I/O device. Therefore, once the translator has set up the program and it has been transmitted to the DIC, the GPC is free to execute other programs. An additional function of the interface is to act as an exit port for the immediate printing or display of DIC problem solutions. Figure 29 is a block diagram of the combined system.

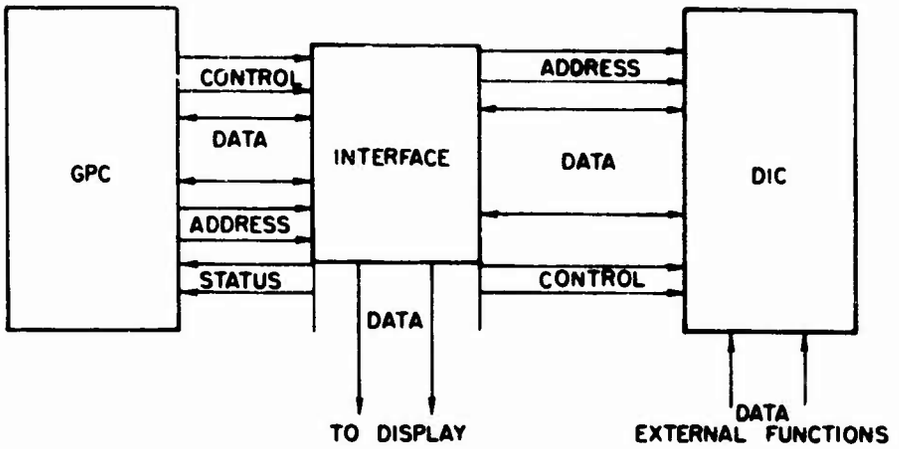


Fig. 29. BLOCK DIAGRAM OF GPC-DIC COMBINED SYSTEM.

Although the DIC generates data at an extremely high rate, it is not necessary to use a DMA (direct memory access) channel since typically the solutions are read out only every hundred or even thousand iterations. The interface will contain a reasonably large temporary storage (512 words) to allow data transfer from the DIC to the GPC via a multi-bit channel under program control, while at the same time allowing immediate display on a CRT or some other device.

H. Computer Simulation

The basic module including the multiplication functions has been simulated on the Stanford Computation Center IBM System 360 model 67. The program was written in FORTRAN, and Fig. 30 is a listing of the simulation. The input data are in essentially the same order and form as they would be for the actual hardware machine, but the initial conditions and some parameters are presented in decimal form. The program execution follows exactly the steps and phases (Section F) of the proposed machine with the exception that the calculations of R2 (the second remainder) and the final output increment are separated to save computing time. The simulation uses the two-loop number system.

After program read-in and verification, all variables and parameters are initialized. The problem solution begins with the first iteration

```
      INTEGER A,B,C,DY,ABY,M,N,DX,SR,TR2,ABR,DZ1,I,K,L,H,ABPR,CNT,COUNT,
     CITER,CYC,F
      INTEGER Y(16),R(16),PY(16),PR(16),UD(16),UDA(16,16),XC(16,3),PMA(1
     C6,2),XA(16,2),DZY(16)
      REAL X,CORRY,D,NORMX,XMAX
      DIMENSION YY(16),MX(16),NORMY(16)
 4000 FORMAT('1','NO DATA FOUND FOR INTEGRATOR #',I3/,' INITIAL CONDITIO
     CNS ARE ASSIGNED TO INTEGRATORS 1 THROUGH ',I3///)
 4010 FORMAT(' ',3X,I6,7X,F7.5,5X,6(F7.5,3X))
 4020 FORMAT(' ',//,'*W A R N I N G ****',//,'Y-VALUE OF INTEGRATOR #',I
     C3,'EXCEEDED THE MAXIMUM VALUE OF',I6/,'DURING ITERATION #',I6/,'TH
     CE VALUE OF Y WAS +,-',I6/,'Y IS RESET TO',I6/,6X,'*****')
 4030 FORMAT(' ','ITERATION',10X,'X',5X,6(4X,'Y(',I2,')',1X))
 4040 FORMAT(' ',3X,'INTE-',2X,'SUM-DY INPUT IS',5X,'DZ-POST-',8X,'DZ GO
     CES TO DY',8X,'DZ GOES TO DX'/,' ',2X,'GRATOR',3X,'SHIFTED UP BY',3
     CX,'MULTIPLICATION',5X,'OF INTEGRATORS',7X,' OF INTEGRATORS',/)
 4050 FORMAT(' ',3X,I3,8X,I4,10X,F10.7,6X,16(I2,','))
 4060 FORMAT('+',68X,I3,',',I3)
COM   READ IN PROGRAM, INITIAL CONDITIONS AND PARAMETERS
      READ(5,*) ITER,XMAX,NORMX,CYC,M,N,COUNT
      CNT=COUNT-1
      DO 10 I=1,16
      UD(I)=0
   10 READ(5,*,END=20,ERR=20) MX(I),Y(I),R(I),PY(I),PR(I),(UDA(I,J),J=1,
     C16),(PMA(I,K),K=1,2),(XA(I,K),K=1,2),(XC(I,L),L=1,3)
   20 WRITE(6,4000)I,I-1
      WRITE(6,4040)
COM   CLEAR AND INITIALIZE DY-STORAGE
COM   PRINT CONNECTION TABLE AND PRE- AND POST-MULTIPLICATION
COM   FACTORS FOR PROGRAM VERIFICATION.
      DO 40 I=1,CYC
      NORMY(I)=M/MX(I)
      K=1
      DO 30 J=1,16
      IF(UDA(I,J).NE.1) GO TO 30
      DZY(K)=J
      K=K+1
   30 DZY(K)=0
      WRITE(6,4050)I,PMA(I,1),PY(I)/(N+0.),(DZY(J),J=1,K)
   40 WRITE(6,4060)(XA(I,J),J=1,2)
      WRITE(6,4030)(I,I=1,CYC)
COM   START ITERATIONS
      DO 120 H=1,ITER
COM   NORMALIZE X AND Y FOR OUTPUT
      DO 50 I=1,CYC
   50 YY(I)=Y(I)/NORMY(I)
      X=(H-1)/NORMX
      IF(X.GT.XMAX) GO TO 120
COM   OUTPUT Y IF CALLED FOR
COM   DELETING THE FOLLOWING STATEMENT CAUSES
COM   ALL Y'S (1-6)TO BE PRINTED : IF(PMA(I,2).EQ.0) GO TO 60
      CNT=CNT+1
      IF(CNT.LT.COUNT) GO TO 60
      WRITE(6,4010)H,X,(YY(I),I=1,CYC)
      CNT=0
   60 CONTINUE
COM   START CYCLE
      DO 110 I=1,CYC
COM   FETCH DY, PRE-MULTIPLY, CLEAR DY-STORAGE
      DZ=0
      DZ1=0
      DY=UD(I)*2**PMA(I,1)
      UD(I)=0
COM   UPDATE Y  AND CHECK FOR Y-OVERFLOW
      Y(I)=Y(I)+DY
      ABY=IABS(Y(I))
      IF(ABY.LE.M) GO TO 70
      Y(I)=ISIGN(ABY-M,Y(I))
      WRITE(6,4020)I,M,H,ABY,Y(I)
   70 CONTINUE
```

Fig. 30. FORTRAN SIMULATION PROGRAM OF DIC MODULE.

```
COM   DETERMINE SOURCE OF DX
      DX=XC(I,2)
      IF(DX.EQ.0) DX=XC(I,1)
      XC(I,1)=0
      A=1
COM   PREDICT INTEGRL. INCREMENT OUTCOME
COM   AND DO POST-MULTIPLICATION, STORE POST-MULT. OUTPUT IN
COM   TEMPORARY STORAGE
      SR=ISIGN(A,R(I))
      TR2=PR(I)+SR*PY(I)
COM   CHECK VALUE OF DX, TERMINATE CYCLE IF DX=0
      IF(DX.EQ.0) GO TO 110
COM   COMPUTE REMAINDER 1 (R) AND INTGRL. OUTPUT  DZ1
COM   DISCARD REMAINDER 2 AND POST-MULTIPL. OUTPUT IF DZ1=0
      R(I)=R(I)+Y(I)*DX
      ABR=IABS(R(I))
      IF(ABR.LE.M) GO TO 80
      R(I)=ISIGN(ABR-M,R(I))
      DZ1=SR
COM   STORE REMAINDER 2, ASSIGN FINAL OUTPUT INCREMENT
      PR(I)=TR2
      ABPR=IABS(TR2)
      IF(ABPR.LE.N) GO TO 80
      PR(I)=ISIGN(ABPR-N,TR2)
      DZ=ISIGN(A,PR(I))
   80 CONTINUE
COM   ROUTE OUTPUT DZ TO DESIRED DY INPUT STORAGE LOCATIONS
      DO 90 K=1,CYC
      C=UOA(I,K)
      IF(C.EQ.0) GO TO 90
      UD(K)=UD(K)+DZ
   90 CONTINUE
COM   ROUTE OUTPUT DZ TO DESIRED DX INPUT STORAGE LOCATIONS
      DO 100 K=1,2
      D=XA(I,K)
      IF(D.EQ.0) GO TO 100
      XC(D,1)=DZ
  100 CONTINUE
  110 CONTINUE
  120 CONTINUE
COM   THE FIRST DATA CARD CONTAINS (SEPERATED BY SPACE OR COMMA) :
COM   ITER   XMAX  NORMX    CYC  M  N  COUNT
COM   THE NEXT DATA CARD MUST BE REPEATED FOR EACH CYCLE  USED
COM   MX(I)   Y(I)  R(I)  PY(I) PR(I)   UDA(,)  PDA(  )   XA(,)  XC(,)
COM   Y,R,PY,PR ARE INTEGER INITIAL CONDITIONS
COM   MX IS A REAL,  THE NEAREST POWER OF 2 SUCH THAT (MX.GE.Y-MAX)
      RETURN
      END
$DATA
$STOP
/*
```

Fig. 30. CONTINUED.

(Loop 120). After each iteration, those integrands that are desired as output are printed provided that the iteration number is a multiple of COUNT (the specified output interval). Loop 110 contains all calculations for each cycle. The computation follows the steps in Section A, again with the exception that part of the post-multiplication operation occurs before the generation of the integral increment and remainder (step 5). The program has been executed with various input data sets, and it performed as predicted with no instabilities.

Chapter IX

CONCLUSION

The goal of this work has been the development of a machine structure that would be useful for the numerical solution of differential equations. Of the various requirements spelled out in Chapter III, the most important consideration has been to achieve a system which would be easy to use and which eventually may lead to an integrated GPC-DIC system, such that the DIC would constitute simply an addition to the GPC processor. The proposed machine has been simulated on a general-purpose computer and the program performed satisfactorily.

A modular computing structure has been introduced, which employs a serial-parallel processing approach. This approach maintains some of the simplicity of communications in serial machines while, at the same time, setting an upper limit on the iteration time regardless of the complexity or size of the problems to be solved. The solution speed of problems that do not require all available modules may be increased by distributing the functions evenly over all modules.

A differential equation is considered to be represented by a number of points in a matrix, where each point designates some integral function whose output must be communicated to other points. This representation leads to two basic methods of inter-module communication: "vertical" and "horizontal" communication. Each of these methods has its advantages; however, because horizontal communication is more appropriate for a wider variety of problems, it has been chosen for the proposed machine.

To eliminate instabilities and oscillations that are inherent in circular number systems and can occur whenever the value is at or near the maximum or minimum, a two-loop number system was introduced. The system has separate positive and negative loops returning to 0 and -1, respectively. Extending the loops such that they overlap but are not identical results in a number system with a hysteresis.

The proposed machine contains within its processor both pre- and post-multiplication of the integral increments. It has been shown that, for the given number system, the outcome of the integrating

cycle can be predicted sufficiently to allow post-multiplication to occur simultaneously with integration. The integral output may be multiplied by either a constant or a variable. Theoretically, the concept of simultaneous integration and post-multiplication can be extended to include any number of post-multiplication factors (both constants and variables).

A simplified method that allows floating-point arithmetic has been introduced, which requires the storing of only a single exponent for each integral cycle. The use of floating-point arithmetic provides dynamic scaling during computations, thereby increasing the accuracy of problem solutions. It also allows the scaling routine contained in the software to be simplified because any inequalities that may occur in the scaling equations can be corrected automatically by an appropriate change of the respective function exponent. This floating-point method does permit simultaneous integration and multiplication.

Hardware construction and testing of at least two DIC modules would be very useful. As experimental information is gathered, the full capabilities of the system can be determined and further improvements may suggest themselves. Future work should be conducted to study the feasibility of hardware incorporation of a unit such as the DIC into general-purpose computers such that the unit would appear as another processor. Further study of the "vertical communication" approach is warranted because, in particular applications, this approach may lead to a superior system.

BIBLIOGRAPHY

Bartee, T. C., Digital Computer Fundamentals, McGraw-Hill Book Co., New York, 1966.

Bartee, T. C., Lebow, I. L., and Reed, I. S., Theory and Design of Digital Machines, McGraw-Hill Book Co., New York, 1962, pp. 252-269.

Bartee, T. C. and Lewis, J. B., "A Digital System for On-Line Studies of Dynamical Systems," Proc. of the Spring Joint Computer Conference, 29, 1966, pp. 105-111.

Bekey, G. A. and Karplus, W. J., Hybrid Computation, John Wiley & Sons, New York, 1968.

Borský, V. and Matyáš, J., Computation by Electronic Analogue Computers, American Elsevier Publishing Co., Inc., New York, 1968.

Bradley, R. E. and Genna, J. F., "Design of a One-Megacycle Iteration Rate DDA," Proc. of the Spring Joint Computer Conference (AFIPS), 21, 1962, pp. 253-364.

Braun, E. L., Digital Computer Design, Academic Press, New York, 1963.

Cunningham, W. J., Introduction to Nonlinear Analysis, McGraw-Hill Book Co., New York, 1958.

Elshoff, J. L. and Hulina, P. T., "The Binary Floating Point DDA," AFIPS Proc. of Fall Joint Computer Conference, 1970, pp. 369-376.

Fenyö, S. and Frey, T., Moderne Mathematische Methoden in der Technik, Birkäuser Verlag, Basel and Stuttgart, 1967.

Forbes, G., Digital Differential Analyzer, Pacoima, California, 1956.

Fowler, M. E., "Numerical Methods for the Synthesis of Linear Control Systems," Automatica, 1, Pergamon Press, London, 1963, pp. 207-225.

Fowler, M. E., "Numerical Methods for Use in the Study of Spacecraft Guidance and Control Systems," Report No. 38.003, IBM Scientific Center, Palo Alto, Calif., 1966.

Gilbert, E. G., "Dynamic Error Analysis of Digital and Combined Analog-Digital Computer Systems," Simulation, 6, 4, Apr 1966, pp. 241-257.

Gill, A., "Systematic Scaling for Digital Differential Analyzers," IRE Trans. on Electronic Computers, EC-8, Dec 1959, pp. 486-489.

Goldman, M. W., "Design of a High Speed DDA," AFIPS Proc. of Fall Joint Computer Conference, 1965, pp. 929-949.

Grabbe, E. M., Ramo, S., and Wooldridge, D. E. (Eds.), Handbook of Automation, Computation, and Control, Vol. 2, "Computers and Data Processing," John Wiley & Sons, New York, 1959.

Gschwind, H., "Digital Differential Analysers," (Ed., P. von Handel), Electronic Computers, Ch. 4, Springer-Verlag, Vienna, 1961.

Hills, F. B., "A Study of Incremental Computation by Difference Equations," Report No. 7849-R-1, Electronics Systems Labs, M.I.T., May 1958.

Hyatt, G. P. and Ohlberg, G., "Electrically Alterable Digital Differential Analyzer," AFIPS Proc. of Spring Joint Computer Conference, 1968, pp. 161-169.

Knudsen, H. K., "The Scaling of Digital Differential Analyzers," IEEE Trans. on Electronic Computers, Aug 1965.

Levine, L., Methods for Solving Engineering Problems Using Analog Computers, McGraw-Hill Book Co., New York, 1964.

Malvino, A. P., "Real-Time Digital Function Generation," Ph.D. dissertation, Stanford University, Stanford, Calif., Sep 1969.

Mayorov, F., Electronic Digital Integrating Computers, American Elsevier Publishing Co., Inc., New York, 1964.

McGhee, R. B. and Nilsen, R. N., "The Extended Resolution Digital Differential Analyzer: A New Computing Structure for Solving Differential Equations," IEEE Trans. on Computers, C-19, Jan 1970, pp. 1-9.

Mitchell, J. M. and Ruhman, S., "The Trice-A High Speed Incremental Computer," IRE National Convention Record, 6, Part 4, 1958, pp. 206-216.

Monroe, A. J., Digital Processes for Sampled Data Systems, John Wiley & Sons, New York, 1962.

Nelson, D. J., "A Foundation for the Analysis of Analog-Oriented Combined Computer System," TR No. 1002-1, Stanford Electronics Laboratories, Stanford, Calif., Apr 1962.

Nelson, D. J., "DDA Error Analysis Using Sampled Data Techniques," AFIPS Proc. of Spring Joint Computer Conference, 1962, pp. 365-373.

Nilsen, R. N., "An Investigation of High Resolution Digital Differential Analyzers," USCEE Report 272, Electronic Sciences Lab., Univ. of Southern California, May 1968.

Owen, P. L., Partridge, M. F., and Sizer, T. R. H., "A Transistor Digital Differential Analyser," Journal British I.R.E., Aug 1961, pp. 83-96.

Peterson, A. M., lecture notes, EE 283, Stanford University, 1968.

Raimondi, A., "Digital Filtering Using Digital Differential Analyzers," Ph.D. dissertation, Stanford University, Stanford, Calif., 1971.

Sage, A. P. and Burt, R. W., "Optimum Design and Error Analysis of Digital Integrators for Discreet System Simulation," Proc. of Fall Joint Computer Conference, 28, 1965, pp. 903-914.

Scarborough, J. B., Numerical Mathematical Analysis, The Johns Hopkins Press, Baltimore, Md., 1966.

Schulz, E. J. and Parasuraman, B., "The Digital Incremental Computer: A New Computing Structure for the Numerical Solution of Differential Equations," Scientific Report No. 36, Stanford Electronics Laboratories, Stanford, Calif., Jun 1971.

Sizer, T. R. H. (Ed.), The Digital Differential Analyser: An Incremental Computer, Chapman and Hall, London, 1968.

Tomović, R., Introduction to Nonlinear Automatic Control Systems, John Wiley & Sons, New York, 1966.

Truitt, T. D., "An Analog-Digital Real-Time Computer," IRE Trans. on Computers, Feb 1962, pp. 46-52.

Wood, P. E., "Digital Differential Analyzers with Arbitrary Stored Interconnections," Professional Group on Electronic Computers, EC-14, 6, Dec 1965, pp. 936-941.

Yu, C.-P., "Digital Filter Design Technique and the Realization of Transfer and Immittance Functions by Using Digital Elements," Scientific Report No. 22, Stanford Electronics Laboratories, Stanford, Calif., Nov 1967.

Yu, C.-P., "Circuit Synthesis Utilizing Digital Variable-Precision-Integrating and Summing Elements," Scientific Report No. 30, Stanford Electronics Laboratories, Stanford, Calif., Dec 1968.