1965

An investigation of a multiple iteration rate incremental data processor

Robert Allan Bruce
Iowa State University

BRUCE, Robert Allan, 1929-
AN INVESTIGATION OF A MULTIPLE ITERATION RATE INCREMENTAL DATA PROCESSOR.

Iowa State University of Science and Technology
Ph.D., 1965
Engineering, electrical

University Microfilms, Inc., Ann Arbor, Michigan

AN INVESTIGATION OF A MULTIPLE ITERATION RATE INCREMENTAL DATA PROCESSOR

by

Robert Allan Bruce

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major Subject: Electrical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

Head of Major Department

Signature was redacted for privacy.

Dean of Graduate College

Iowa State University
Of Science and Technology
Ames, Iowa

1965

TABLE OF CONTENTS

I. INTRODUCTION

With the expansion of requirements for real-time physical data accumulation and processing, and with the development of reliable, precise and high-speed digital hardware, a special-purpose digital processor which would be used onboard moving vehicles, at remote data pick-up points or in conjunction with large-scale data processors is feasible and desirable.

Often, much of the physical data to be accumulated or generated is analog in nature and the functional variation as well as the computational processing desired can be defined by differential equations for many applications. With this in mind, the usage of the computational processing of a high-speed digital differential analyzer (DDA) appears most applicable. Furthermore, the control functions required in a DDA for computation are very compatible with those required for a wide variety of data accumulation. The inherent building block form of the DDA also allows compatible design for each application.

A. Past Applications of Incremental Processors

The digital differential analyzer evolved as an outgrowth of the analog differential analyzer.

The first useful analog differential analyzer was developed by V. Bush and A. H. Caldwell (1) and used electro-mechanical

ball and disk integrators. Application of this type of analyzer was primarily devoted to laboratory simulation.

Later development of the voltage operational amplifier (2) made feasible the development of electronic analog computers, such as the Reeves REAC and Electronic Associates, Inc. PACE. Such machines are also used for laboratory simulation of control systems. The basic computer elements consisting of integrators and amplifiers are used, however, in many control systems to provide system compensation.

The first digital differential analyzer was developed by Northrup Aircraft, Inc. in 1950 to realize greater accuracy and reliability. This computer, designated MADDIDA (3), utilized a magnetic drum for storage and a serial arithmetic unit. This computer and others, such as the Computer Research Corporation CRC 105 (4) and the Honeywell 256 - integrator DDA (5), were used primarily for navigation equation solution.

The DDA has been incorporated with a general-purpose computer to provide a total navigation, guidance and monitoring computer for military airborne and missileborne application. The Autonetics VERDAN (6) and Litton Industries C-900 (7) computers are typical of this type of combination computer.

The above DDA's used magnetic drum or disk storage and a serial-arithmetic unit commonly shared with memory storage blocks to implement multiple integrator operation. This time-sharing operation limits the iteration frequency to 400

iterations per second or below. This range of iteration rate restricts the application of these computers.

Machines which exhibited a significant increase in iteration rate were developed by Packard Bell Corporation in the TRICE computer (8) and by Hazeltine Technical Development Center in the SPEDAC computer (9). Each of these machines used a separate arithmetic unit and storage in the form of a delay line or set of flip-flop registers for each separate computing module. The computing modules consisted of several types; a digital integrator, a constant multiplier, a variable multiplier, etc. This approach provides a significant increase in iteration rates to as high as 1 megacycle. The associated increase in hardware causes an accompanying increase in cost and causes reduction in reliability which again severely limits application.

B. Extension of Applications by Multiple-Rate Processing and Minimal Memory

Some of the applications of an incremental data processor such as the DDA which appear feasible and desirable are itemized below.

1. Real-time physical data accumulation and computation based on solution of differential equations:
   a. Synthesis of transfer functions for control systems such as inertial navigators, celestial trackers, adaptive autopilots and armament control.

b. Collection and processing of data for recording or transmission such as data pick-up and filtering, squaring, correlating, etc. to reduce the amount of data to be stored or transmitted.

c. Computational operations as required for navigation such as dead reckoning, celestial triangle solution, etc.

2. Synthesis of differential equations for simulation of a physical problem:

a. Linear, integral or differential mathematical equation solution.

b. Control system analysis.

c. Correlation and smoothing studies.

d. Faster than real-time computation for prediction or decision making such as flight path or trajectory prediction, orbit prediction, fuel monitoring, etc.

The characteristics desired in an incremental processor for these applications are presented in the following text.

The processor should be easily programmable in terms of the physical problem and this program should be easily modified with little processor changes. The use of the DDA integrator function is readily interpretable in terms of the differential equations of the physical problem and the use of a stored program will allow proper modification.

The processor memory should be non-volatile; i.e., immune

to loss of program and data in the event of power shutdown or failure. This characteristic requires the use of one of several possible magnetic storage devices: a magnetic drum, magnetic disk, magnetic tape, a magnetic core array or a thin magnetic film array. The magnetic core array or thin-magnetic-film array are considered most practical to attain high-speed, small-space requirements and high reliability which are not inherent in the other magnetic storage devices.

The amount of electronic hardware necessary to drive and sense the signals in a magnetic array is proportional to the amount of storage required. A significant emphasis is then placed on minimization of the amount of this storage in the use of such an array.

In all incremental machines developed to date using a time-shared arithmetic unit, a common iteration frequency has been used for all integrators and this frequency has been limited to from 50 cps to 400 cps by hardware limitations.

A wide variation of operating frequency ranges exist in various portions of many control systems. Such is the case in inertial systems, for example, where the gyro stabilization loops and the accelerometer output processing function must be sampled at a rate up to 1 kcps while the basic Schuler loops and gyro-compassing functions need only be sampled at rates of 0.1 cps or less. This is similarly true for tracking systems where both a fast loop and a slow outer loop are synthesized.

A similar situation exists in data accumulation such as in airborne flight-data recording where fuel monitoring need only be sampled over fractions of seconds or less while navigation-position data must be handled at a much faster rate.

The use of a common iteration rate for all processing integrators, therefore, constrains some of the integrators to be processed at a much higher rate than is required and, alternately, restricts the iteration rate for others to a rate which is unusable in many applications.

The use of multiple-iteration rates for sets of DDA integrators would circumvent the above limitations and provide a significant improvement in processor performance. An improvement in the iteration rate by an order of magnitude or greater above the basic rate for a small portion of the total number of integrators is feasible. This would increase the iteration rate for those integrators to within the range of 1 kcps to 20 kcps which would allow usage in carrier-frequency analog systems or in audio-frequency data processing. This higher rate could alternately be used effectively to improve the computational precision in time-dependent equation solution as a result of the associated reduction in time increment.

In summary, specific characteristics of an incremental processor which would extend its useful application are a basic DDA integrator functional operation, an easily determined stored program, minimal storage in the form of a magnetic array and multiple-iteration rates for sets of the integrators.

II. DESIRED INCREMENTAL PROCESSOR CHARACTERISTICS

A. Basic Computation and Input Accumulation

Before the incremental processor characteristics are studied, a discussion of the basic DDA integration process is in order.

The integration of $y(x)$, as shown in Figure 1, with respect to x over the limits x_o to x_m is given in Equation 1.

$$z = \int_{x_o}^{x_m} y(x)\, dx \tag{1}$$

This integration can be approximated by Equation 2 for increments of x equal to constant Δx.

$$z \simeq \sum_{i=1}^{m} y_i(x)\, \Delta x = \Delta x \sum_{i=1}^{m} y_i(x) \quad \text{where} \quad \frac{x_m - x_o}{m} = \Delta x \tag{2}$$

Similarly, $y(x)$ can be approximated by Equation 3.

$$y(x) \simeq \sum_{i=1}^{m} (\Delta\, y(x))_i + y_o \tag{3}$$

The variables z and y can be adequately approximated by proper choice of the incremental size or quantization of x and y (i.e., Δx and Δy). The increments, Δx and Δy, are physically represented in the digital machine by one ternary bit having possible values of +1, 0 or -1 and the summation consists of normal binary addition.

In a physical problem, y will not exceed some finite maximum, y_{max}. The sum z will not, therefore, change more than

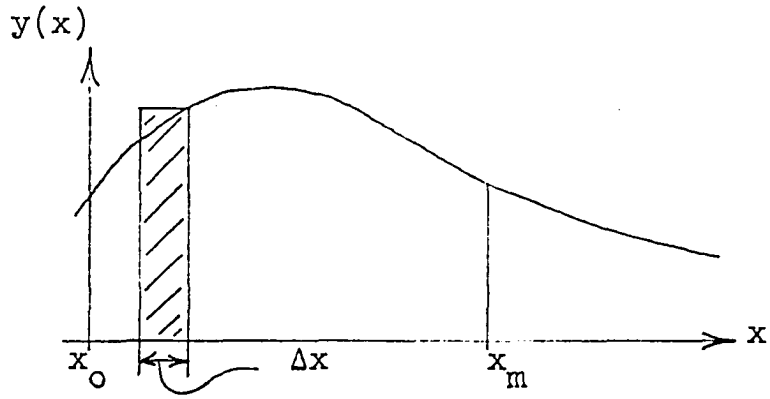


Figure 1. Basic integration

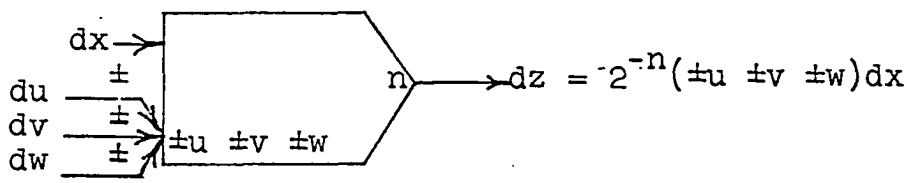


Figure 2. DDA integrator symbol

y_{max} for any one of the ith summations with Δx normalized to ± 1 or 0 as above. With y_{max} physically limited to a value less than n bits in length and with Δx normalized to 1, the summation of y_{max} and z can be performed by adding y to only the lowest n bits of z and by applying the nth carry bit to the (n+1)th bit of z. By outputting the carry bit to the (n+1)th bit at each iteration and retaining the remaining lowest n bits, referred to as the remainder, r_n, for summation with y at the next iteration, a quantization of z can be effected. That is, accumulation of the (n+1)th bit output at each iteration will yield a value proportional to z as in Equation 4 and truncated at the nth bit.

$$w = 2^n z \quad \text{where } w = \sum_i y_i(x)\ \Delta x \text{ truncated} \tag{4}$$

$$z = 2^{-n} w = \sum_i 2^{-n} y_i\ \Delta x \text{ truncated} \tag{5}$$

The accumulation of the (n+1)th bit of z can be considered to be the accumulation of an increment of z, Δz. The increment of z can then be represented as in Equation 6.

$$\Delta z = 2^{-n} y \Delta x \quad \text{or} \quad dz \simeq 2^{-n} y dx \tag{6}$$

The incremental output Δz can then be stored and applied as either a dependent-variable or independent-variable input to any other integrator, and, in fact, as an input to the present integrator at the next iteration time.

Each integrator is then processed sequentially in one iteration period.

The variables x, y, and z discussed thus far have been considered digital variables represented by a set of binary bits normally called a "word". The conversion from or to analog variables can be performed by the use of a proportionality constant, which, in a given equipment, is determined by the specific operation of the analog-to-digital converters or digital-to-analog converters. This is represented as in Equation 7.

$$x_d = k_x x_a \quad \text{or} \quad dx_d = k_x dx_a \qquad (7)$$

where x_d = digital variable
x_a = analog variable
k_x = proportionality constant with units of pulses per unit analog variable

Greater versatility in use of the integration process is achieved by performing accumulation of several incremental inputs (i.e., performing an intermediate algebraic summation) into a given integrator as in Equation 8.

$$\Delta y = \pm \Delta u \pm \Delta v \pm \Delta w \pm .... \qquad (8)$$

The symbolic representation for each integrator adopted by the author is shown in Figure 2.

The basic digital operations required are listed below.

1) Extract from memory Δu, Δv, Δw from Δz storage
2) Sum Δu, Δv, Δw to form Δy total = Δy_t
3) Extract y_{n-1} and add to Δy_t to form y_n
4) Store y_n into memory

5) Multiply y_n by Δx to form $y_n \Delta x$

6) Extract r_{n-1} and add to $y_n \Delta x$ to form r_i and Δz

7) Store r_i and Δz into memory

These operations are shown diagrammatically in Figure 3.

Two forms of external input conversion and/or accumulation which utilize the basic integrator operations are shown in Figures 4 and 5.

The first form consists of: a digital-to-analog voltage conversion of the updated variable, y_n; comparison of this resultant voltage with the input voltage, v_{input}; and amplification and pulse shaping of the resultant difference to provide an incremental output. This is shown in Figure 4.

The second form of input accumulation is utilized to generate the increment of a digital input, u, which is available as one word. The updated variable, y_n, is digitally compared bit-by-bit with the input, u, to determine a positive or negative difference. This difference then provides the incremental output. This form is shown in Figure 5.

In each of the above forms of input accumulation, the increment accumulation is identical with that used in an integrator performing internal computations. This similarity allows use of a set of integrators for either computation or sequential input sampling and conversion or combinations of both.

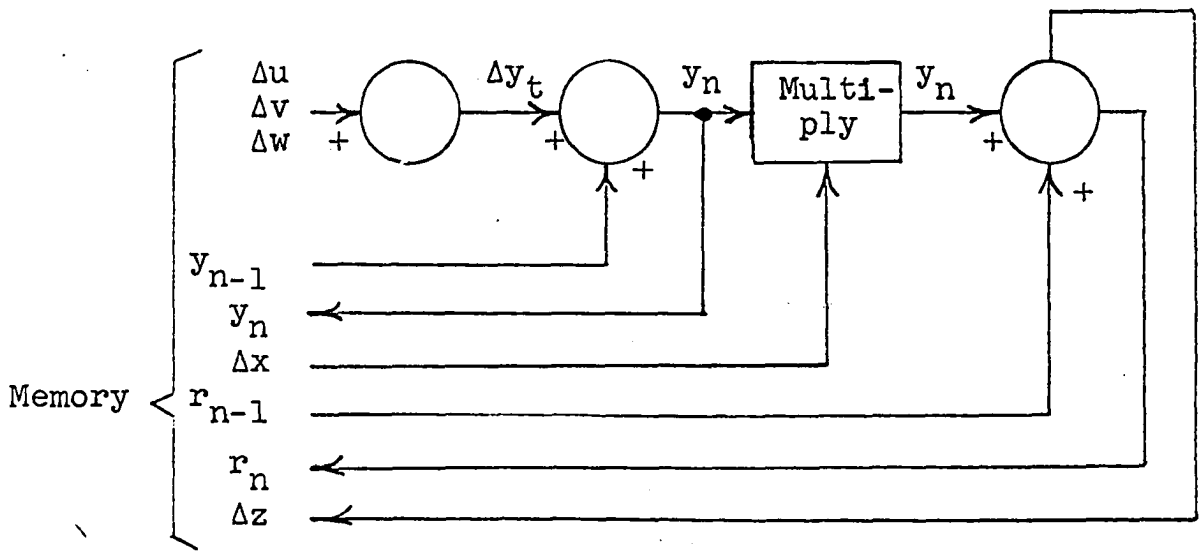


Figure 3. Basic arithmetic operations

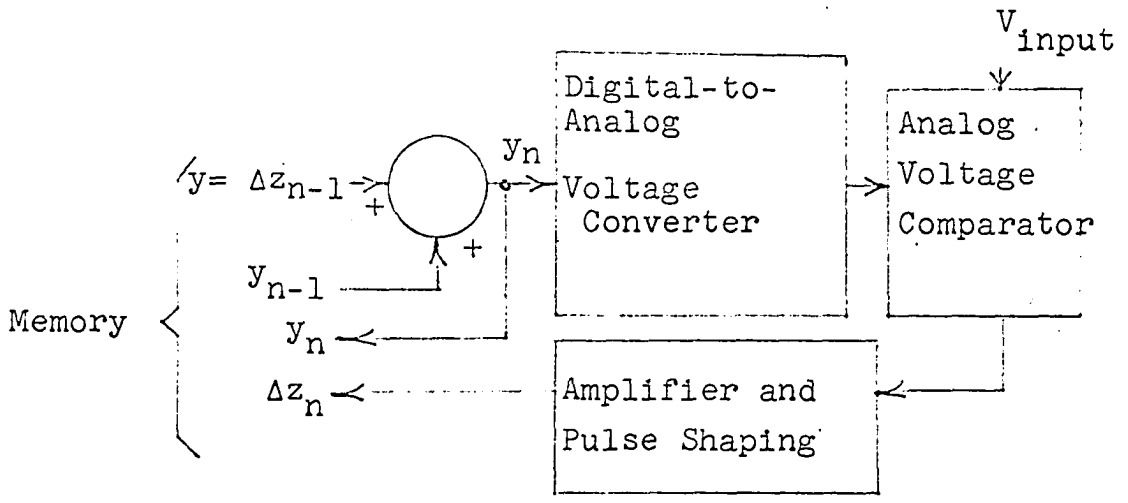


Figure 4. Voltage converter

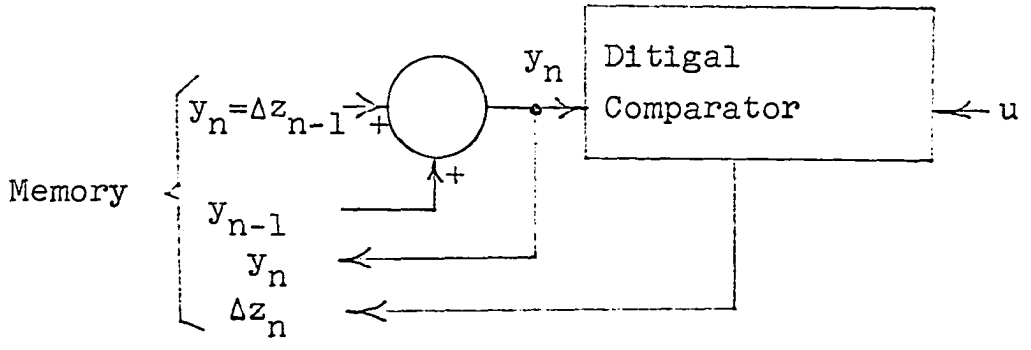


Figure 5. Input accumulator

B. Minimization of Memory

In reviewing the computational processing presented in section II. A memory space must be available for data storage of three digital variables for each integrator; i.e., variables Δz, y and r. The variable Δz requires 2 bits of data to specify the one ternary bit of incremental data. The variables y and r require the same number of at least n bits and n will vary for each specific integrator.

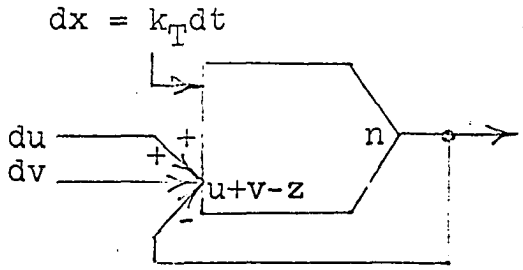
To more thoroughly understand the range of n desired, several commonly used forms of integrator connections are presented in Figures 6, 7, 8 and 9.

Additionally, the relationship between n binary-bit accuracy and m decimal-digit accuracy is indicated in Equation 9.

$$n = \log_2 2^n = \log_2 10^m = \log_2 10 \log_{10} 10^m = 3.32m \quad (9)$$

For example, 6 digit accuracy requires 20 bit accuracy and 8 digit accuracy requires 27 bit accuracy.

The accuracy of sine-cosine generation, constant multiplication, variable multiplication and time integration is dependent on the usage of the incremental processor. For laboratory synthesis of sets of differential equations entirely within the processor, 4 to 8 digit (14 to 27 bits) accuracy may be required. However, in real-time processing of data, analog-to-digital or digital-to-analog conversion of the data limit the accuracy to the order of 3 to 4 digits (10 to 14



$$dz = (u+v-z)\ 2^{-n} k_T dt$$

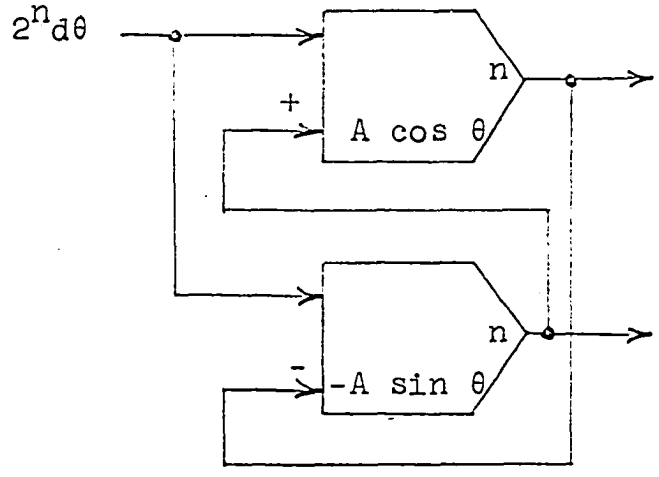
For $\frac{dz}{dt} \ll (u+v-z)\ 2^{-n} k_T$

$$u + v \simeq z$$

or $d(u + v) \simeq dz$

or $du + dv \simeq dz$

Figure 6. Summer



$$dz = 2^n\ 2^{-n} A \cos\theta\ d\theta$$

$$= d(A \sin\theta)$$

$$dz = -2^n\ 2^{-n}\ A \sin\theta$$

$$= d(A \cos\theta)$$

$$(A < 2^n)$$

Figure 7. Sine-cosine generator

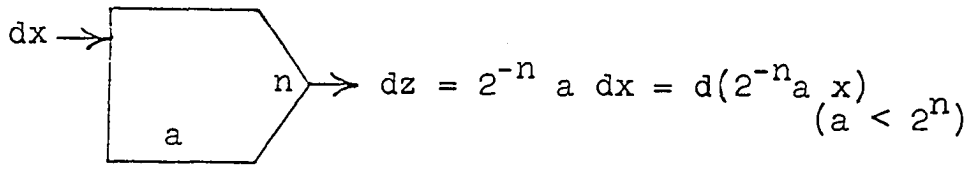


Figure 8. Constant multiplier

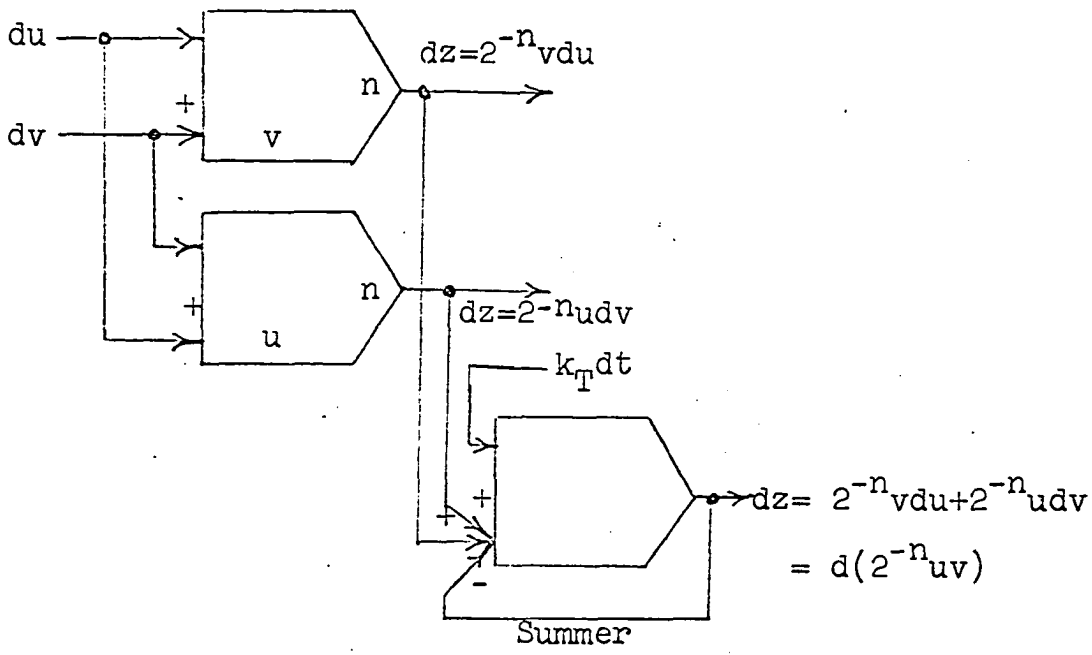


Figure 9. Variable multiplier

bits). For the summer, n need only be large enough to allow storage of 2^n-1 bits where n is typically 3 or 4.

Typical values for n and an expected average for n are given in Table 1.

Table 1. Typical integrator bit lengths

Integrator function	Bit length n
Summer	3
Sine-cosine generator	10 to 14 $\simeq$ 12
Constant multiplier	10 to 14 $\simeq$ 12
Variable multiplier	10 to 14 $\simeq$ 12
Time integrator	10 to 14 $\simeq$ 12
Total	51
Average $n = \frac{51}{5} \simeq 10$	

With these expected conditions of a possible maximum n of 27 bits and an average n of 10 bits, variable-bit-length storage is necessary for the r and y data in order to minimize the memory requirements. An additional 5 bits of memory are required, however, for each integrator to uniquely specify n between 1 and 28 bits.

In addition to the Δz, r, y and n size data, instruction data must be stored to specify the inputs to each integrator. This data must specify the type of input (dx, dt, +dy, -dy) and the location of the Δz data to be applied as the input.

The number of inputs for various integrator functions are given in Table 2.

Table 2. Number of inputs per integrator function

Integrator function		Number of inputs
Summer		4
Sine-cosine generator	Integ. # 1	2
	Integ. # 2	2
Constant multiplier		1
Variable multiplier	Integ. # 1	2
	Integ. # 2	2
Time integrator		2
Low pass filter	Integ. # 1	3
	Integ. # 2	1
Total		19

Average number of inputs $= \frac{19}{9} \simeq 2$

A condition exists, therefore, where the maximum number of inputs is 4 or 5 for the summing function, for example, and the average number of inputs is 2. Memory reduction can then be accomplished by establishing control or instruction to accumulate a variable number of inputs. An additional 3 bits of memory are required per integrator to specify the number of inputs if up to 7 inputs are provided for.

Each integrator input can then be specified with a minimum of 2 bits to identify the type of input and $\log_2 N$ bits to uniquely identify the integrator output (Δz) to be used as the input where N is the nearest power of 2 equal to or greater than the total number of integrators.

By use of variable length storage for n, a variable number of inputs and a binary coded specification for each input, a minimum amount of memory per integrator is achieved. The number of bits of storage per integrator for specific data

is as follows: 3 bits for specifying the number of inputs; 5 bits for specifying the r, y computation bit length; 2n bits of r, y data; 2 bits of Δz store; 2m bits to define type of input for m inputs and m $\log_2$ N bits to specify the input to be used. The total number of bits of memory minimum, B_{min}, per integrator is given by

$$B_{min} = 10+2n+m(2+\log_2 N) \qquad (10)$$

For the expected average values of n=10 and m=2, B_{min} is given by

$$B_{min} = 34 + 2 \log_2 N \qquad (11)$$

For comparative purposes, the three existing methods of data storage used in DDA synthesis will be discussed along with the memory requirement for each.

The first method utilizes a magnetic drum for storage, such as in the Northrup Aircraft, Inc. Maddida computer (3), Autonetics Verdan computer (6) and Minneapolis-Honeywell 256 Integrator DDA (5), where the data is stored in several tracks or channels on the drum as indicated in Figure 10. The number of bits per integrator for specific data is as follows: $2n_{max}$ bits of r, y data where n_{max} is the maximum allowable integrator bit length; 2 bits of Δz store; $2m_{max}$ bits to specify the type of input where m_{max} is the maximum number of inputs and $m_{max} \log_2 N$ bits to specify the input to be used. The corresponding number of total bits of memory for drum storage per integrator, B_{MD}, is given by

$$B_{MD} = 2 + 2n_{max} + m_{max} (2 + \log_2 N) \qquad (12)$$

For $m_{max} = 5$ and $n_{max} = 24$, the total drum storage per

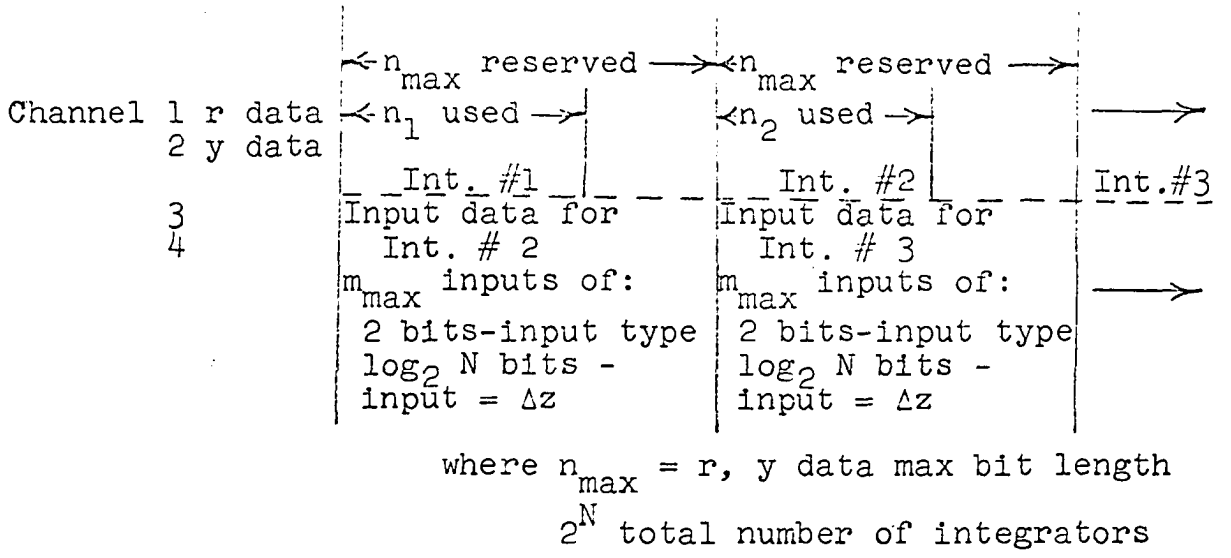


Figure 10. Magnetic-drum data format

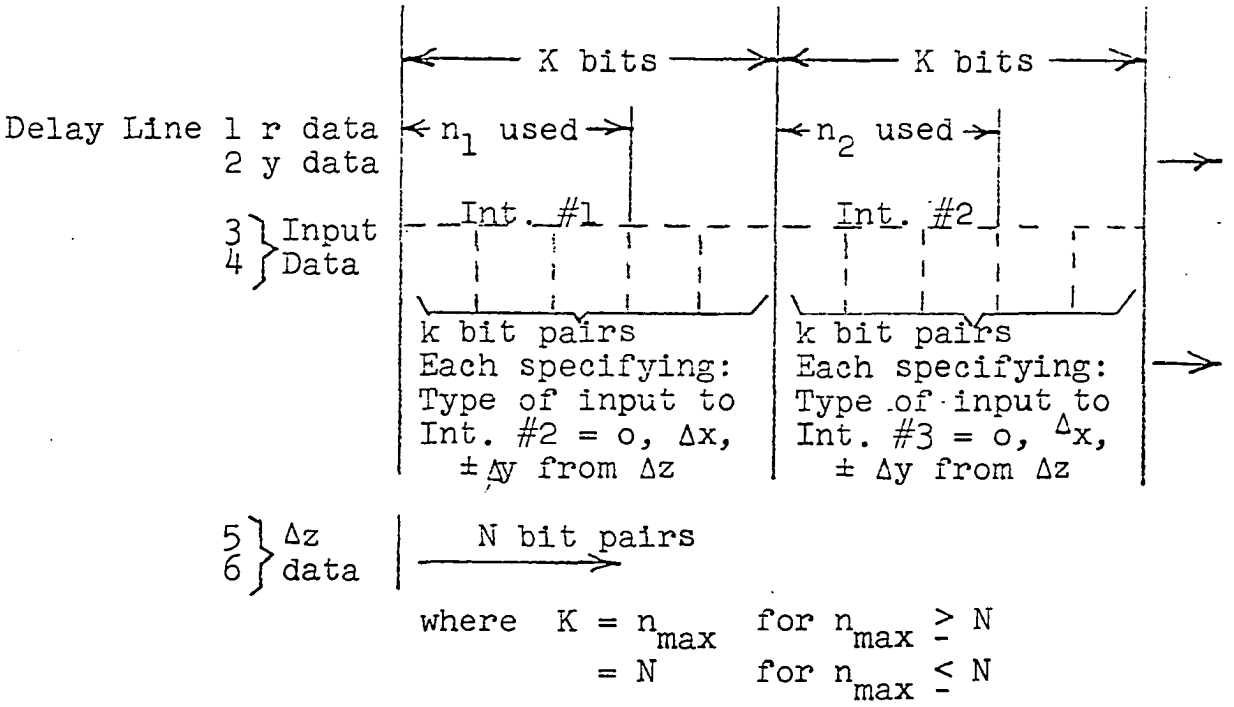


Figure 11. Delay-line data format

integrator is given as

$$B_{MD} = 60 + 5 \log_2 N \qquad (13)$$

The second method utilizes recirculating delay lines, such as in the Computer Control Co., Inc., SPEC computer (10), in a format similar to that used in drum storage. The data is distributed down six delay lines at any instant of time as indicated in Figure 11. The total number of bits of memory per integrator for delay-line storage, B_{DL}, is given in Equation 14 where the maximum integrator bit length is n_{max} and the maximum number of integrators is 2^N.

$$B_{DL} = \begin{cases} 2 + 4\, n_{max} & \text{for } n_{max} \geq N \\ 2 + 4\, N & \text{for } n_{max} \leq N \end{cases} \qquad (14)$$

For $n_{max} = 24$, the total delay line storage per integrator is given by

$$B_{DL} = \begin{cases} 98 & \text{for } 24 \geq N \\ 2 + 4N & \text{for } 24 \leq N \end{cases} \qquad (15)$$

The third method consists of programming a general-purpose computer to synthesize the DDA integrator function. A typical program to perform this function is given in Table 3, tabulated below.

Table 3. GP program of DDA integrator function

Instruction	Memory requirement
1) Clear and add Δy_1	1 word - instruction
2) Add Δy_2	1 word - instruction

Table 3 (Continued)

Instruction			Memory requirement
3)	Add	y_{n-1}	1 word - instruction 1 word - y_n data
4)	Store	y_n	1 word - instruction
5)	Multiply	Δx	1 word - instruction
6)	Add	r_{n-1}	1 word - instruction 1 word - r_n data
7)	Store	r_n	1 word - instruction
8)	Sense overflow and store	Δz	1 word - instruction 1 word - Δz data
Total			10 words

For a word length of 24 bits, the total number of bits of memory for GP synthesis per integrator, B_{GP}, is 240 bits.

The measure of reduction of memory by use of the minimal memory DDA technique is given by the ratios of B_{MD}, B_{DL} and B_{GP} to B_{min} as in Equations 16, 17 and 18 respectively.

$$\frac{B_{MD}}{B_{min}} = \frac{60 + 5 \log_2 N}{34 + 2 \log_2 N} \tag{16}$$

$$\frac{B_{DL}}{B_{min}} = \begin{cases} \dfrac{98}{34 + 2 \log_2 N} & \text{for } 24 \geq N \\ \dfrac{2 + 4N}{34 + 2 \log_2 N} & \text{for } 24 \leq N \end{cases} \tag{17}$$

$$\frac{B_{GP}}{B_{min}} = \frac{240}{34 + 2 \log_2 N} \tag{18}$$

These results are graphed in Figure 12 as a function of N.

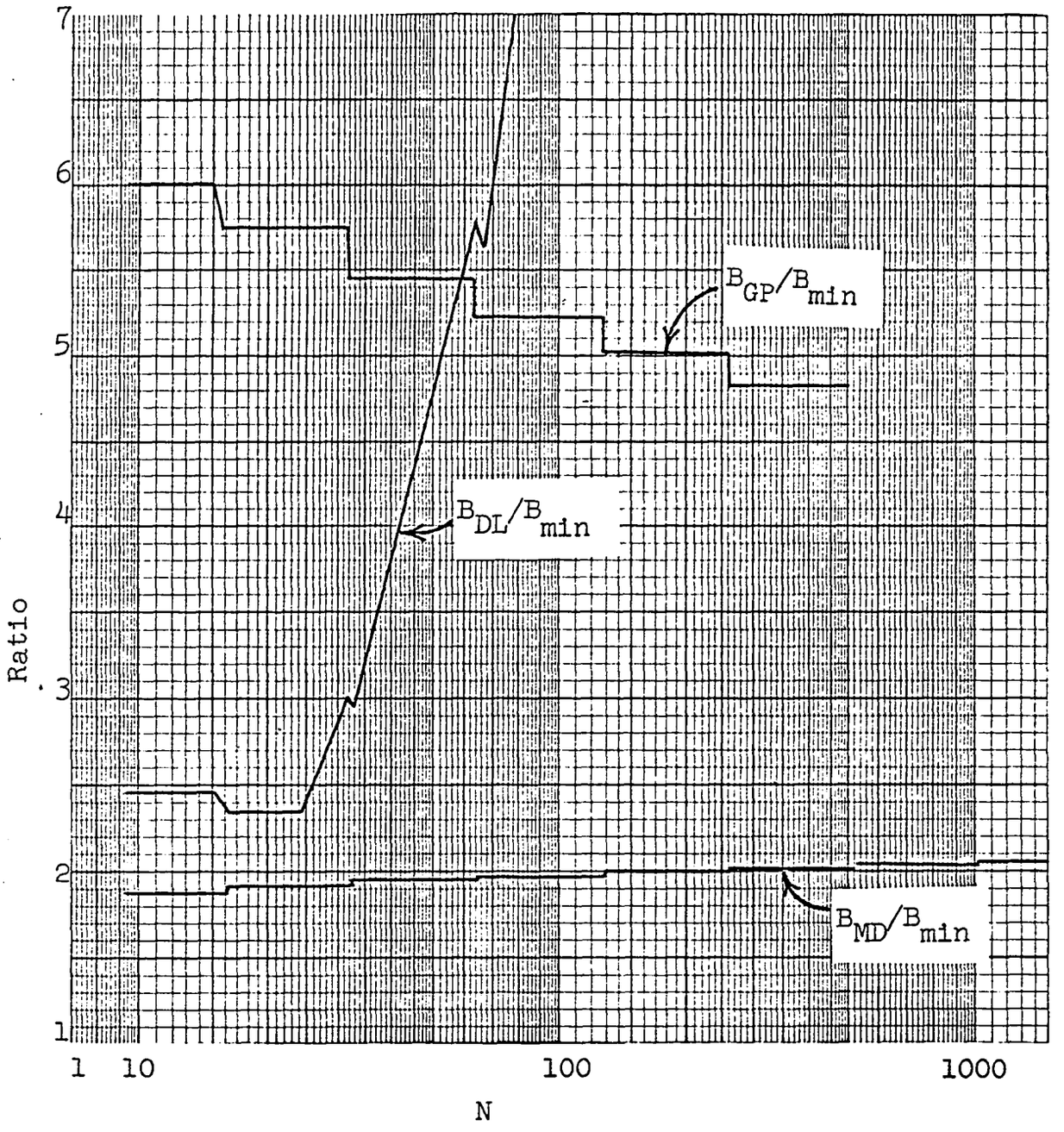


Figure 12. Storage requirement ratios

C. Multiple-Rate Processing

The timing sequence to perform the DDA integrator ditial operations presented in section II. A is illustrated in Figure 13.

The timing sequence presented in Figure 13 is repeated for each iteration period.

The functions performed during each specific memory cycle in Figure 13 are given in Table 4.

Table 4. Integrator processing cycle functions

Memory cycle	Function performed
L	Extract n value and number of inputs
I_1	Extract Δz number as 1st input and 1st input type
ZE_1	Extract 1st input Δz value Add to cleared Δy accumulator if Δy input Temporarily store if Δx input
I_2	Extract Δz number as 2nd input and 2nd input type
ZE_2	Extract 2nd input Δz value Add to Δy accumulator if Δy input Temporarily store if Δx input
$D_n, \ldots\ldots, D_s$	Extract r, y (nth lowest bit to sign bit) Add y_n to Δy lowest bit to update y_n Multiply resultant y_n by Δx Add r_n to resultant $y_n \Delta x$ to update r_n Insert undated y_n into memory Insert undated r_n into memory

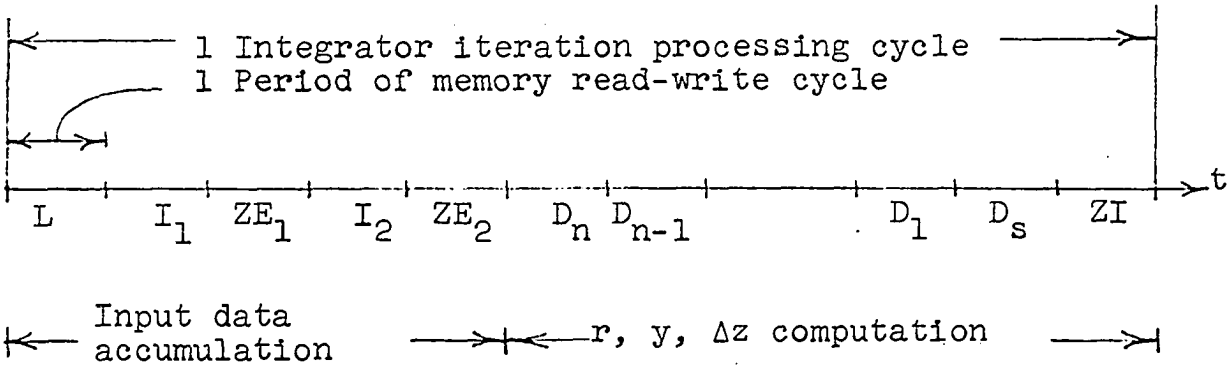
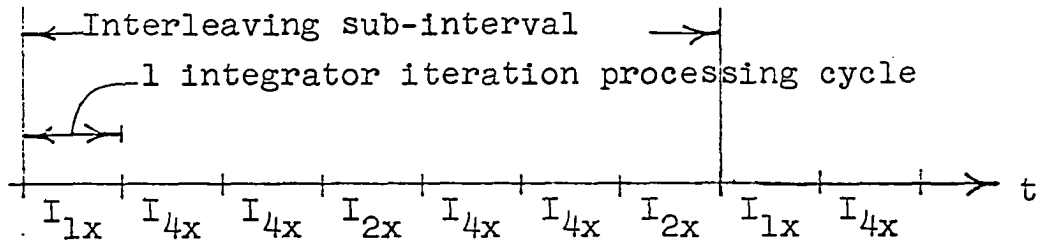


Figure 13. Control timing sequence



where I_{1x} = 1 of the "1x" integrators
I_{2x} = 1 of the "2x" integrators
I_{4x} = 1 of the "4x" integrators

Figure 14. Interleaving sequence

Table 4 (Continued)

Memory cycle	Function performed
	Repeat above for n-1st bit through sign bit, D_s Sense overflow, Δz, and temporarily store at D_s time
ZI	Insert temporarily stored Δz output into memory

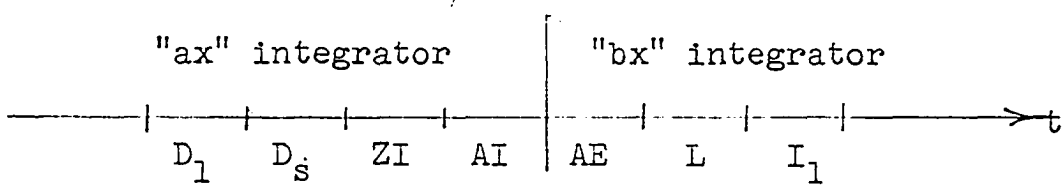
Each integrator in most existing incremental computers is processed sequentially once per iteration period and the input data for each integrator is accumulated during the time that the prior integrator r and y data are being processed. This time overlapping of data for two different integrators cannot be performed when multiple iteration rates are used, however, and the input and computational data must be stored as an addressable closed set of data in memory.

In order to perform multiple iteration rates with different sets of integrators, the interleaving of process cycles for the various integrators must be considered. This interleaving as illustrated in Figure 14 for the case of the fundamental iteration rate (1x) and multiples of 2 (2x) and 4 times (4x) the fundamental rate with the same number of integrators at each rate. Each of the set of 1x, 2x and 4x integrators would be processed sequentially to complete the fundamental iteration period.

The sample sequence in Figure 14 indicates the need for random memory access to any integrator block of data and clarifies the need for a closed set of data per integrator. The memory addressing for the interleaving sequence then consists of random access to the starting address of each data block and normal sequential addressing within the data block.

The addressing then uses normal sequential addressing within the data block, sequential processing of the set of integrators of a given iteration rate and switching between the sets of integrators of the different iteration rates according to the interleaving sequence. In the process of address transferring between the sets, the last address for each of the iteration sets not in process is temporarily stored and returned to upon continuing the processing of a given set. This temporary storage and transfer requires added control timing within the integrator timing sequence as indicated in Figure 15.

The iteration periods for the various rates should now be considered to determine the proper interleaving sequence. For the case where the number of integrators operating at each of the various rates is different, the fundamental recurrence period of processing will be a multiple of the average 1x period. This multiple, k_1, can be determined by considering the total number of integrator processor cycles



where AI = temporary store of last "ax" set address

AE = Extract from temporary storage last "bx" set address

Figure 15. Control transfer sequence

in the fundamental recurrence period, T_f, for 3 iteration rates as given in Equation 19. The terms $k_1 dp$ and $d_1 ep$ must be multiples of

$$T_f = k_1 (p + dp + ep)\, T_c = (k_1\, p + k_1\, dp + k_1\, ep)\, T_c \quad (19)$$

where p = total number of "1x" integrators
d = total number of "ax" integrators in each interleaving sub-interval
e = total number of "bx" integrators in each interleaving sub-interval
T_c = average integrator cycle period

the total number of ax integrators, q, and bx integrators, s, respectively as indicated in Equations 20 and 21.

$$k_1 dp = k_2 q \quad (20)$$

where k_1, k_2 and k_3 are lowest integers

$$k_1 ep = k_3 s \quad (21)$$

The resultant average iteration periods for 1x, ax and bx iteration periods are $T_{1x_{av}}$, $T_{ax_{av}}$ and $T_{bx_{av}}$ as given in Equations 22, 23 and 24 respectively.

$$T_{1x_{av}} = \frac{T_f}{k_1} = (1 + d + e)\, pT_c \quad (22)$$

$$T_{ax_{av}} = \frac{q}{dp}\, T_{1x_{av}} = \frac{1 + d + e}{d}\, qT_c \quad (23)$$

$$T_{bx_{av}} = \frac{s}{ep}\, T_{1x_{av}} = \frac{1 + d + e}{e}\, sT_c \quad (24)$$

The above indicate that integral multiples of iteration rates by use of the interleaving technique are possible only for integral ratios of q and s to p. Non-integral multiples and a $T_{1x_{av}}$ dependence on p, d, e and q can be accounted for

in actual computation, however, by use of a constant-multiplier interconnection operating on the time variable and by precise measurement of the 1x iteration period.

The iteration periods will vary around the average values due to variation in T_c, due to relative values of p, q and s and due to the specific sequence within the interleaving sub-interval. The variation due to T_c and the relative values of p, q and s will tend to be small due to averaging over several process cycles. The specific sequence within the interleaving sub-interval will be the dominant cause of instantaneous variations in iteration periods and should be considered in further detail.

An adequate upper bound on the variation can be achieved by use of the following technique to determine the interleaving sequence.

Step 1. Given p, q, s and an approximate value for T_c T_{1x}, T_{ax}, T_{bx} required, determine d and e from Equations 22, 23 and 24 to satisfy the iteration period requirement.

Step 2. Determine k from Equation 25 where k is an integer.

$$\frac{e}{d} - 1 \leq k \leq \frac{e}{d} \qquad (25)$$

Step 3. Specify the sequence from the above determined d, e and k as illustrated in Figure 16.

The maximum relative variation, v_{ax} and v_{bx}, for the ax

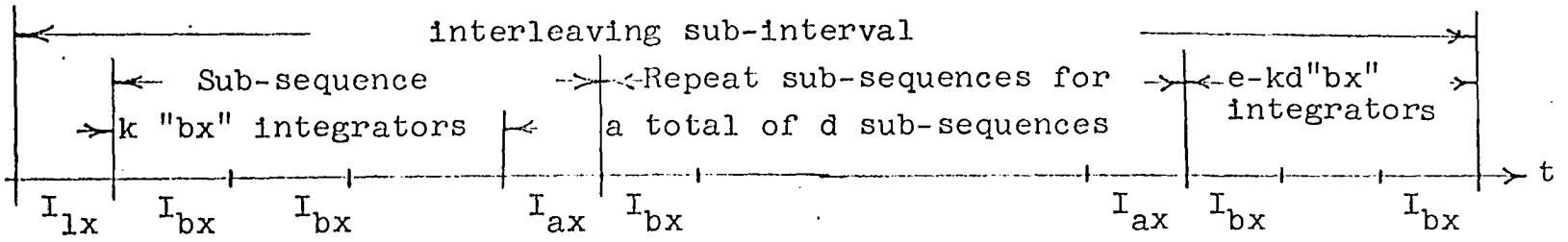


Figure 16. Interleaving sub-sequence

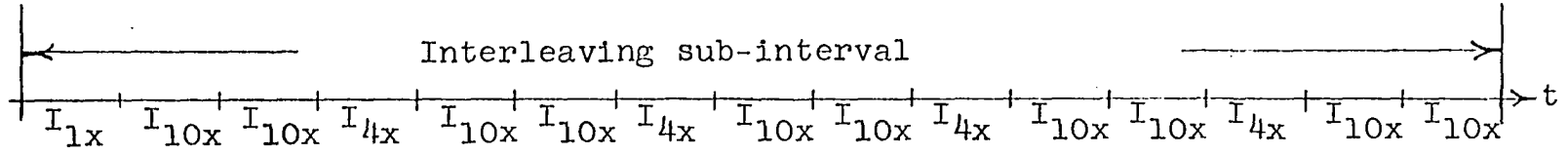


Figure 17. 1x, 4x and 10x interleaving sequence

and bx iteration periods using the above technique are given in Equations 26 and 27 respectively. Less than 20% maximum

$$v_{ax} \leq \frac{T_{ax\ max\ variation}}{T_{bx_{av}}} = \frac{eTc}{dT_{bx_{av}}} = \frac{e}{(1 + d + e)q} \qquad (26)$$

$$v_{bx} \leq \frac{T_{bx\ max\ variation}}{T_{bx_{av}}} = \frac{Tc}{T_{bx_{av}}} = \frac{e}{(1 + d + e)s} \qquad (27)$$

variations is achieved for the number of ax and bx integrators (q and s) equal to or greater than 5. In performing computations, the average iteration period must be known precisely but a 20% variation over several periods will have a secondary effect on computational accuracy.

An example of the interleaving sequence using the above technique for d = 4 and e = 10 is illustrated in Figure 17.

Additionally, the iteration rates for 4 combinations of p, q, s, d and e are given in Table 5. A value of T_c = 9 μ sec. was used in the computations which is an expected value if a thin-magnetic-film memory were used.

Table 5. Iteration rate combinations

	Total no. of integ. p+q+s	No. of lx integ. p	No. of ax integ. q	No. of bx integ. s	f_{lx}	f_{ax}	f_{bx}
d=4 e=10	50	30	12	8	246.7 cps	2.467 kcps	9.252 kcps
	100	70	20	10	105.7 cps	1.480 kcps	7.402 kcps
	200	140	40	20	52.9 cps	740 cps	3.701 kcps
d=0 e=0	250	250	-	-	444.4 cps	-	-

III. IMPLEMENTATION OF DESIRED PROCESSOR CHARACTERISTICS

A. Basic Operation

The specific characteristics of the incremental processor to be implemented were chosen based on the sample problem to be performed.

The sample problem was chosen to illustrate the minimal memory and multiple-iteration-rate techniques. The three speeds of 1x, 4x and 10x were chosen to illustrate the multiple rates. A problem requiring 5 to 8 integrators operating at each iteration rate was assumed and it was decided to run the same problem at each iteration rate.

The specific problem chosen consists of generating sine and cosine functions.

Two programs were chosen to be run to illustrate the effect of the multiple-iteration rates.

The first program consists of cycling sine and cosine for many cycles to illustrate the gross characteristics of the generated sinusoidal functions. The integrator interconnections or mapping (11, 12) for each iteration rate for the first program are shown in Figure 18.

The second program consists of rotating the angle, ωT, incrementally and recording the sinusoid amplitude values over two segments of a single period. The intent in running this program is to illustrate the fine-grain structure effect of multiple rates. The mapping for this second program for each

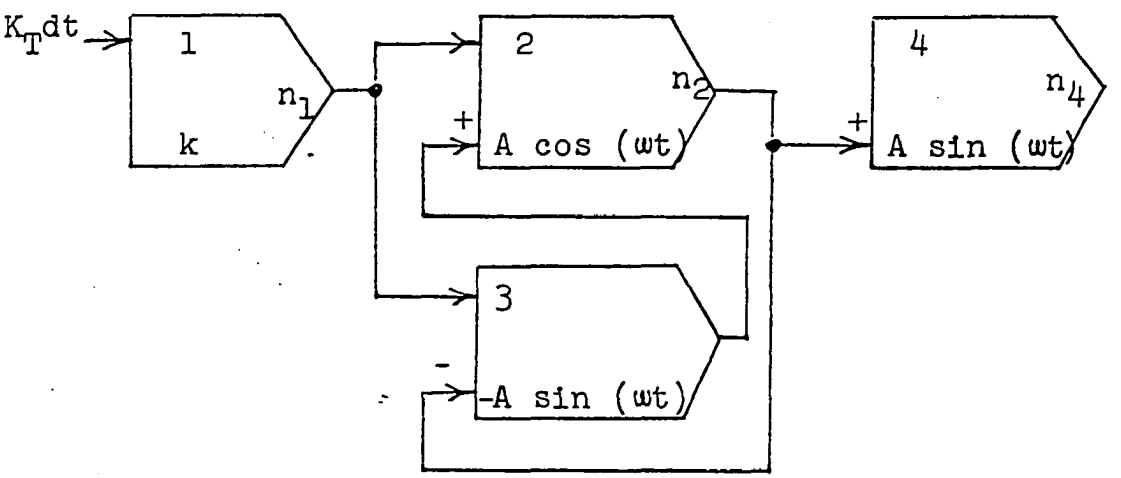


Figure 18. Sine/cosine cycle test map

rate is shown in Figure 19.

The integrator scaling for each program was to be so chosen that the angular frequency, ω, for each of the 1x, 4x and 10x integrator sets is the same.

Based on the sample problem and the general characteristics discussed in section II.A, the incremental processor characteristics are specified as given in Table 6.

Table 6. Incremental processor characteristics

Characteristic	Data
Maximum number of integrators	32
Maximum number of inputs per integrator	7
Maximum integrator bit length	30
Multiple iteration rates	1x, 4x, 10x
Memory requirement	128 words 8 bits per word
D/A converter channels	3

The basic control timing sequence for each integrator processing cycle as shown in Figure 13 was utilized in the processor with two exceptions.

The first exception was the inclusion of an additional control code, designated SOC (for Special Operating Condition). This control code was generated to provide an indication that the specific integrator in process is the last integrator of

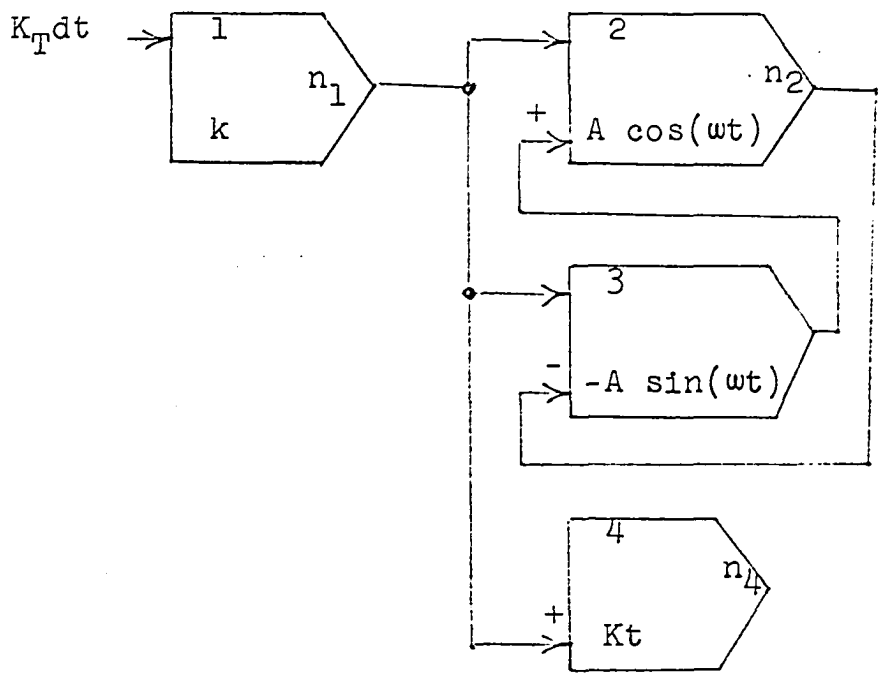


Figure 19. Sine/cosine rotation test

a specific iteration rate and/or that the y data of the specific integrator in process is to be outputted to the digital-to-analog converter.

The second exception was the change from serial arithmetic to a combination parallel-serial arithmetic. This parallel-serial arithmetic consists of performing binary arithmetic 4 bits in parallel for both the y and the r data and performing the 4 bit sets in serial to complete the handling of all n bits of the y and r data. This change was made to achieve a significant iteration rate increase through use of the 8 bit parallel output characteristic of the memory.

With the above data in mind, the allotment of data in memory as available at the memory output during the existence of each of the control codes is given in Table 7.

Table 7. Memory program data allotment

Memory Word Address	Control Code	Memory Bit Number	Data Content	Comments
m	L	1 through 5	integrator bit length	binary coded
		6 through 8	number of inputs	binary coded
m+1	I_n	1 through 5	number of integrator whose output is to be used as the input	binary coded
		6 through 7	type of input	coded as in Table 8.
m+2	I_{n-1}	(Same as for I_n)		

Table 7 (Continued)

Memory Word Address	Control Code	Memory Bit Number	Data Content	Comments
	. . . .			
m+n	I_1	(1 through 7	same as for I_n)	
		8	existence of SOC code	
m+n+1	SOC	2, 3	output to D/A converter	coded as in Table 9.
		5	last integrator of present iteration rate	
m+n+2	D_p	1 through 4	least significant bits of y	y data in binary 2's complement
		5 through 8	least significant bits of r	r data in binary 2's complement
m+n+2	D_{p-1}	1 through 4	next 4 significant bits of y	
		5 through 8	next 4 significant bits of r	
	. . . .			
m+n+1+p	D_1	1 through 3	3 most significant bits of y	
		4	y sign bit	
		5 through 7	3 most significant bits of y	
		8	r sign bit	

The logic code for integrator input type is given in Table 8.

Table 8. Logic code of itegrator input type

Input type	Memory Bit Logic State (I_n code) Bit 7	Bit 6
Δt	0	0
$+\Delta y$	0	1
$-\Delta y$	1	1
Δx	1	0

The logic code for an output to the D/A converter and the multiplexer channel to which the output is applied is given in Table 9.

Table 9. Converter and multiplexer channel code

Data content	Memory Bit Logic State (SOC code) Bit 3	Bit 2
Output to converter/multiplexer channel 1	1	0
Output to converter/multiplexer channel 2	0	1
Output to converter/multiplexer channel 3	1	1

The functional block diagram of the incremental processor utilizing the program coded output and variable data from memory is shown in Figure 20.

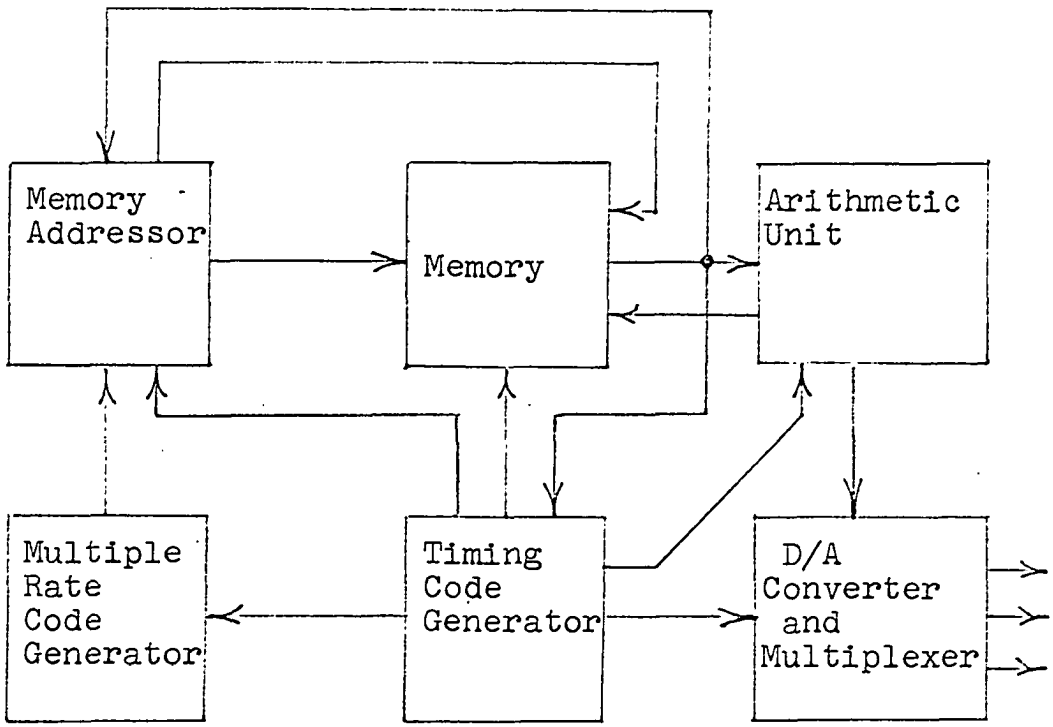


Figure 20. Incremental processor block diagram

The detailed operation of those functions shown in Figure 20 is discussed in sections III. B through III. F. The detailed operations of these functions are discussed in terms of basic logic functional units such as counters, storage registers, logic-decoding gates, etc. The detailed logic making up these basic logic functions consists of normal combinational and sequential logic circuits (13, 14) and has been omitted from the main body of the report. A detailed discussion of the logic is presented in the appendix.

B. Memory Addressor

The block diagram of the memory addressor is shown in Figure 21.

The PIZ counter contains the address of the Δz data of the integrator presently in process. This counter is advanced at the beginning of an integrator process cycle (L code time) and the counter data is selected to address memory when this integrator's Δz data is inserted into memory (ZI code time). When changing from one iteration-rate integrator to the next, the past integrator Δz location is inserted into memory and the next integrator Δz location is inserted into the PIZ counter from memory (at AI and AE code times).

The DA counter contains the addresses of the program instructions, y data and r data. This counter is advanced during these instruction extraction times (L, I_n, SOC code times) and during the y, r data computation times (D_n code

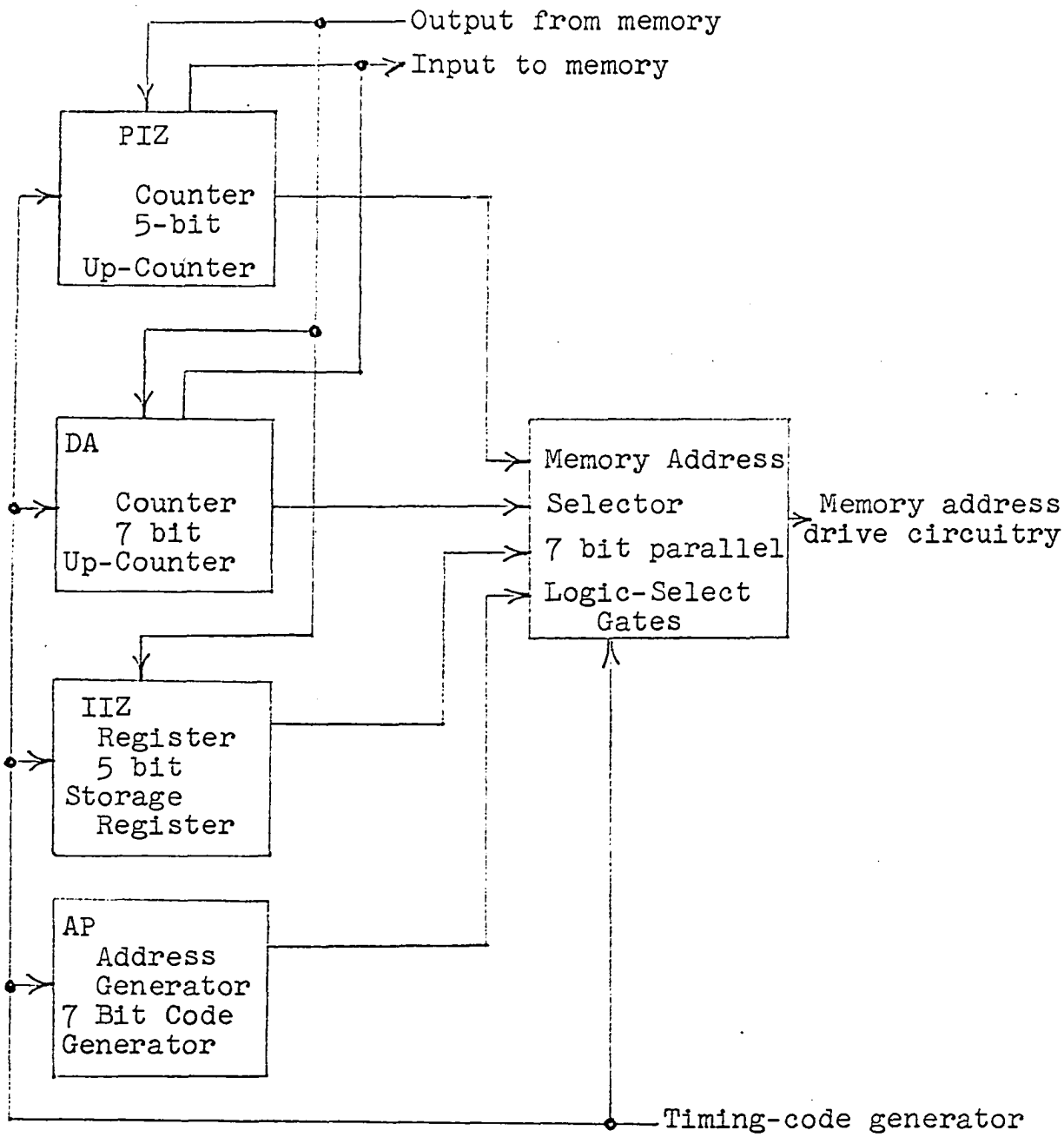


Figure 21. Memory addressor

times). The DA counter is selected to address memory at these times also. Similarly to the PIZ counter, counter data is interchanged with memory data when changing iteration rate integrators (at AI and AE code times).

The IIZ storage register is set by memory output data to the location of the Δz data of the integrator to be used as an input at the input definition time (I_n code time). This counter is selected to address memory then at this input extract time (ZE_n code time).

Specific locations in memory are allotted for storing a specific integrator's initial instruction address and also its Δz address for each of the three iteration rates. These locations are used for temporarily storing the address of the last completed instruction of the program sequence for one iteration-rate set of integrators while processing integrators in a second set. The AP code generator generates the proper address code for one of the above allotted storage locations during the iteration rate change times (AI and AE code times).

C. Timing Generator

The block diagram of the timing generator is shown in Figure 22.

The timing control codes for one complete integrator-process cycle are generated by the timing generator. These timing codes are illustrated in Figure 13 with the additional inclusion of the SOC time code.

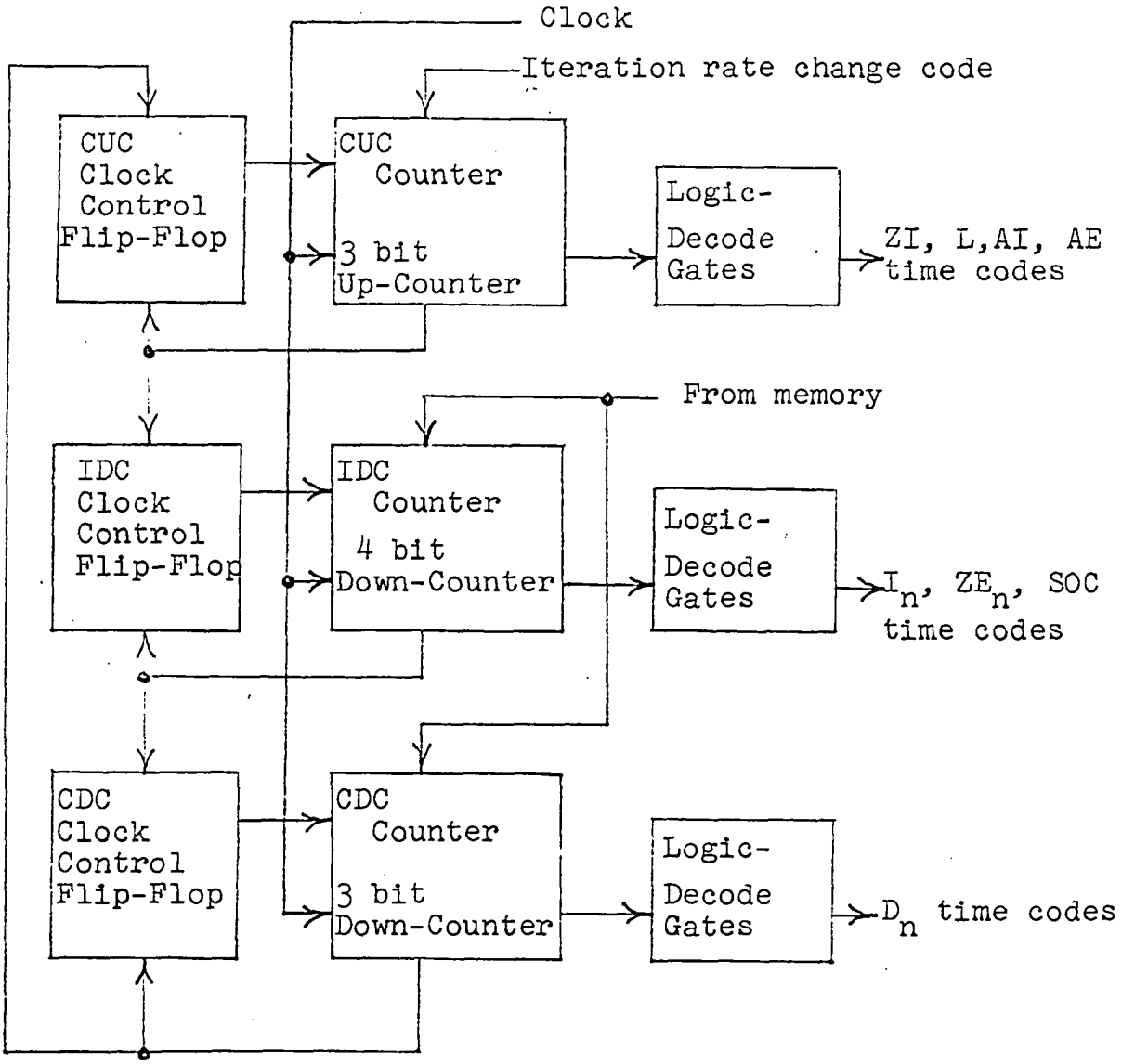


Figure 22. Timing generator

The clock control of the counters cycles sequentially from the CUC counter to the IDC counter to the CDC counter and back to the CUC counter. This is performed by using the reset to one clock control flip-flop to provide the set control for the next.

The CUC counter generates the length data code (L time code), the Δz insertion to memory code (ZI time code) and the address interchange codes (AI and AE time codes). The L and ZI codes are generated each cycle and the AI/AE codes are generated on those cycles in which an iteration rate change exists.

The IDC counter generates the input definition codes (I_n time codes), the Δz extract codes (ZE_n time codes) and the special condition code (SOC time code). This is performed by setting the counter with the memory data containing the number of inputs at the length code time and counting down.

When the IDC clock control is set, the count down is to zero for absence of an SOC code and to -1 for presence of an SOC code. The presence or absence of the SOC code is sensed on memory bit 8 at I_1 time.

The CDC counter generates the y, r data computation codes (D_n time codes). This is performed similarly to the IDC counter whereby the CDC counter is set by the bit length data present in the memory output at L time. Count down then occurs when the CDC clock control flip-flop is set.

D. Multiple-Rate Code Generation

The multiple-rate code generator provides control codes indicating the iteration rate of the integrator in process (1x, 4x and 10x control codes) and the presence or absence of the need for an iteration rate change at the completion of the integrator process cycle presently being performed.

These codes are generated by use of a 4-bit up-counter which is advanced at the beginning of each integrator process cycle and by use of logic decoding gates to determine the sequence as shown in Figure 17.

E. Arithmetic Unit

The block diagram of the arithmetic unit is shown in Figure 23.

The arithmetic unit performs the basic arithmetic operations using the y, r, Δy and Δx data as discussed in section II.A.and as illustrated in Figure 3. The y and r data form in memory and in the arithmetic unit is binary 2's complement. Each Δy input and the Δx input consists of one ternary bit of data.

The accumulation of incremental inputs, Δy and Δx, is performed successively by extracting control data from memory defining the type of input (at I_n code time) and then by extracting from memory the Δz data to be used as an incremental input (at ZE_n code time).

The ΔI Data Type Store as shown in Figure 23 is set by

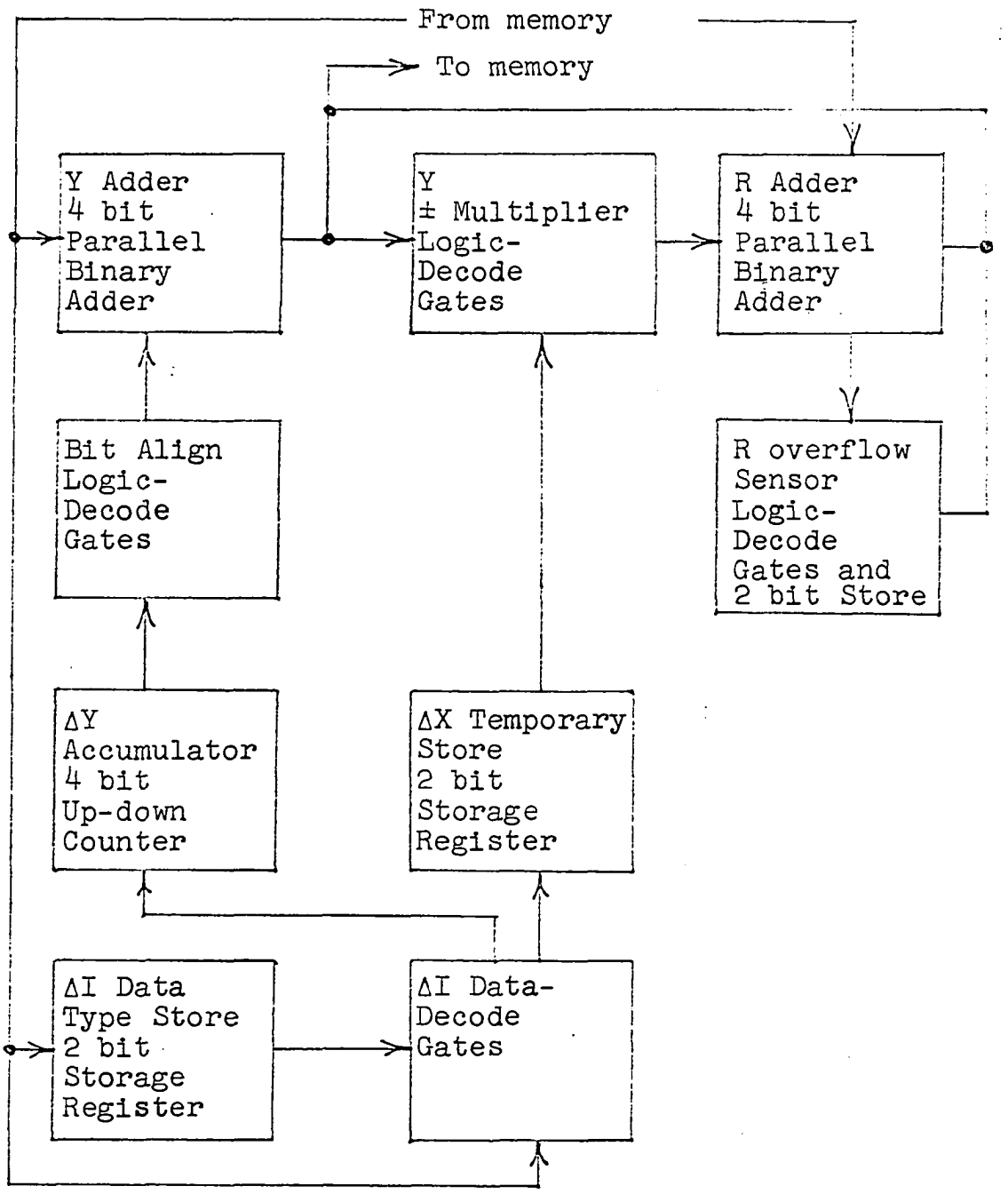


Figure 23. Arithmetic unit

the memory data defining the input type at each I_n code time. The ΔI Data-Decode Gates use this data to decode and transfer the input increment from memory at the following ZE code time (ZE_n code time).

The ΔX data output from the ΔI Data-Decode Gates is stored temporarily in the ΔX Temporary Store for use during the y, r computation times (D_n code times). This data consists of the inputted Δz ternary bit for a dx input and a positive ternary "one" for a dt input.

The ΔY data output from the ΔI Data-Decode Gates is applied as an input to the ΔY Accumulator. A positive Δz ternary bit input is applied to the up-count input and a negative Δz ternary bit input is applied to the down-count input. The total ΔY accumulation is then retained in the counter for use at the y, r computation time.

The least-significant bit of y as extracted from memory at D_n time can exist in any one of four memory output y bits (bits 1 through 4). The ΔY accumulation must then be aligned for application to the Y Adder. This is performed in the Bit Align Logic-Decode Gates by use of the 2 least-significant bits in the integrator bit length data stored at L code time.

The Y Adder performs parallel binary addition at each of the y, r computation times using the 4 bits of y data from memory and the Bit Align Logic outputs. The four bit sets of data are then added serially by use of storage of the most

significant carry bit for use at the next D time. The resultant y data is inserted into memory and also applied to the Y ± 1 Multiplier.

The Y ± 1 Multiplier uses the ΔX Temporary Store data to transfer y directly to the R Adder for Δx being a positive increment and transferring the 2's complement of y to the R Adder for Δx being a negative increment.

The R Adder performs binary addition of the Y ± 1 Multiplier output and the r data from memory in the same way as performed in the Y Adder. The resultant r data is then inserted back into memory.

The R Overflow Sensor detects and temporarily stores a Δz overflow of the R Adder output data. This sensing is performed by logic decoding of the states of the r sign bit and the r carry bit into the sign bit. A positive Δz ternary bit is stored for positive overflow and a negative Δz ternary bit is stored for negative overflow. This stored data is then inserted into memory at the Δz insertion time (ZI code time).

F. Memory Input-Output

The memory input-output functions are shown in Figure 24.

The memory-word drivers use the Memory-Address Selector outputs discussed in section III. B to drive the proper memory word line being addressed.

The memory-sense-line outputs at word-drive time are amplified in the memory-sense amplifiers and inputted to set the

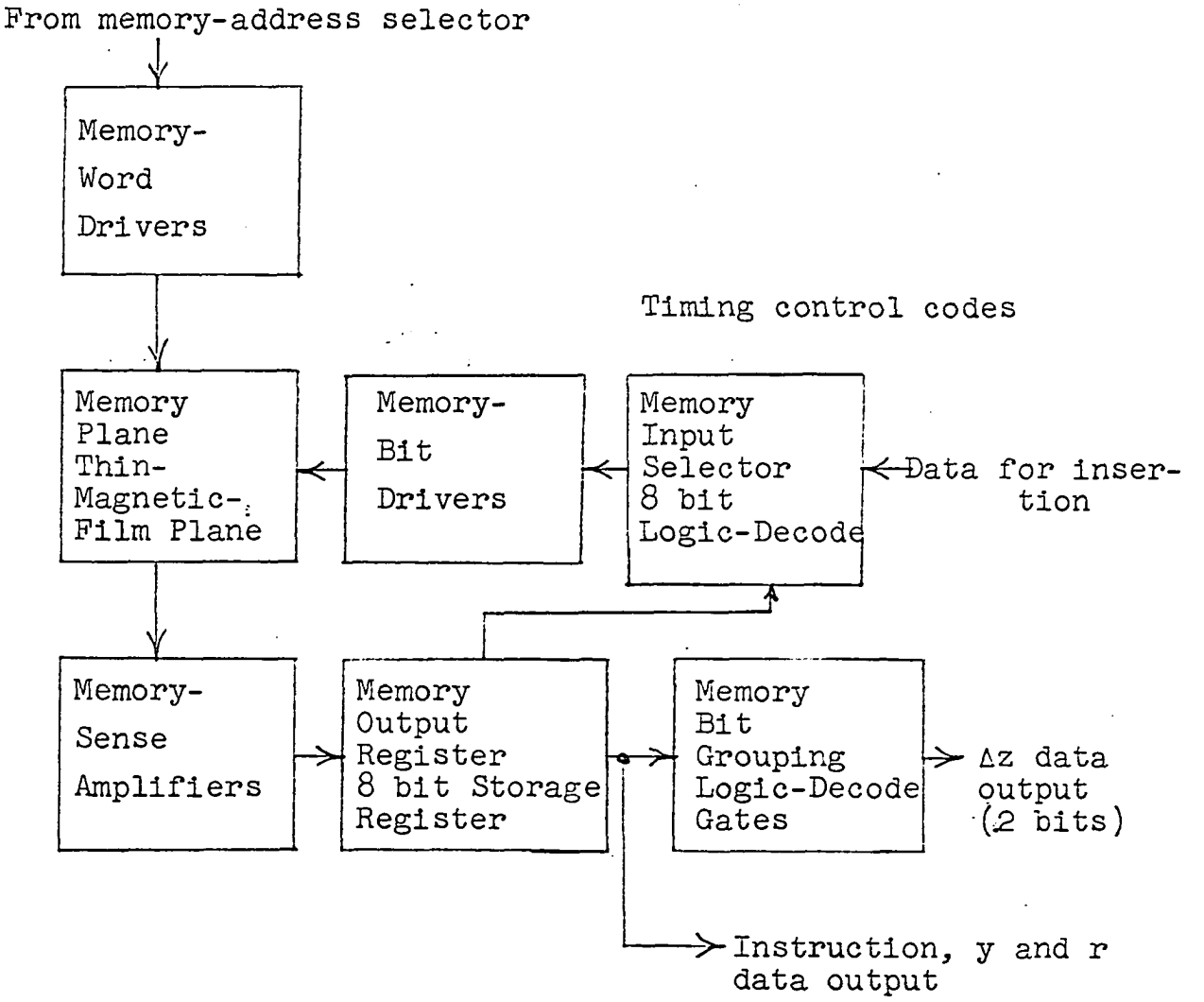


Figure 24. Memory input-output

Memory-Output Register. This data is then retained in the Memory Output Register until the initiation of the next memory-access cycle. The Memory Output Register data provides then the instruction data, y and r data, for use within the various sections of the incremental processor.

The Memory Bit Grouping Logic selects the proper 2 bits of the 8 bits available from memory output to supply the Δz ternary-bit output at ZE code time. The control data used to provide the proper selection is available within the 2 least-significant bits retained in the IIZ storage register at ZE time.

The Memory Input Selector selects the various data to be inserted into memory such as y and r data and including re-inserting memory-output data to allow retention of the program. The basic integrator-timing codes are used to control the selection.

The Memory Input Selector output is applied to the memory-bit drivers. These drivers in turn control the final state of stored data in the thin-magnetic-film plane word being addressed.

IV. SAMPLE PROBLEM SYNTHESIS

The sample problem to be synthesized has been defined in basic terms in section III. A. The mappings of integrator interconnections for the sample problem are shown in Figures 18 and 19.

Scaling of the various integrators must now be done to specifically define the equation solution. This scaling as discussed in Mendelson (4) and Braun (15) consists of determining the physical scaling constants and each integrator bit length. The scaling is done by use of the relationship between differentials as given in Equations 6 and 7.

In the following analysis, the iteration-rate constant will be referred to as K_T iterations per second and the integrator bit length will be referred to as n_m for integrator number m as defined in Figures 18 and 19.

The pulse or increment rate input to the time scaler, integrator number 1, is given in Equation 28.

$$\frac{\Delta x_1}{\Delta t} = K_T \tag{28}$$

With the initial value of the y register for the time scaler set to k, the incremental output of the time scaler is given in Equation 29.

$$\Delta z_1 = 2^{-n_1} k \Delta x_1 = 2^{-n_1} k K_T \Delta t \tag{29}$$

The incremental rate output of the time scaler is given by Equation 30.

$$\frac{\Delta z_1}{\Delta t} = 2^{-n1}\ k\ K_T \tag{30}$$

The Δy accumulation to generate y consists of accumulating the pulses as they are present at a given pulse rate at the y input. The accumulation then performs the time integration of the pulse rate input.

Defining the time scaler output pulse rate as K_t, the incremental output rates and outputs of integrator number 2 and 3 are given in Equations 31 through 34.

$$\frac{\Delta z_2}{\Delta t} = K_t\ 2^{-n2}\ z_3 \tag{31}$$

$$\Delta z_2 = K_t\ 2^{-n2}\ z_3 \Delta t \tag{32}$$

$$\frac{\Delta z_3}{\Delta t} = -K_t\ 2^{-n3}\ z_2 \tag{33}$$

$$\Delta z_3 = -K_t\ 2^{-n3}\ z_2\ \Delta t \tag{34}$$

By setting n_2 equal to n_3, Equations 31 through 34 approximate those for sine and cosine as given in Equations 35 through 38 respectively.

$$\frac{\Delta z_2}{\Delta t} = \frac{\Delta[A \sin (K_t\ 2^{-n2} t)]}{\Delta t} = K_2\ 2^{-n2}\ A \cos (K_t\ 2^{-n2}\ t) \tag{35}$$

$$\Delta z_2 = \Delta[A \sin (K_t\ 2^{-n2}\ t)] = K_t\ 2^{-n2}\ A \cos (K_t\ 2^{-n2}\ t)\Delta t \tag{36}$$

$$\frac{\Delta z_3}{\Delta t} = \frac{\Delta[A \cos (K_t\ 2^{-n2}\ t)]}{\Delta t} = -K_t\ 2^{-n2}\ A \sin (K_t\ 2^{-n2}\ t) \tag{37}$$

$$\Delta z_3 = \Delta[A \cos (K_t\ 2^{-n2}\ t)] = -K_t\ 2^{-n2}\ A \sin (K_t\ 2^{-n2}\ t)\Delta t \tag{38}$$

The magnitude A is specified by the initial values of

the y registers for integrator numbers 2 and 3.

The angular frequency, ω, is specified as in Equation 39.

$$\omega = k_t \, 2^{-n_2} = \frac{\Delta\theta}{\Delta t} \tag{39}$$

The angular rotation per input time pulse is given by dividing the angular frequency as given in Equation 39 by Δx_1 as given in Equation 28. The resultant rotation per input time pulse, $\Delta\theta/\Delta x_1$, is given in Equation 40.

$$\frac{\Delta\theta}{\Delta x_1} = \frac{\omega\Delta t}{k_T \Delta t} = 2^{-n_2} \text{ radians} \tag{40}$$

The accumulation of the Δy input to integrator number 4 approximates the time integral of the input incremental rate as given in Equation 41.

$$y_4 \simeq \int_o^t \frac{d(y_4)}{dt} \, dt \tag{41}$$

The digital-to-analog converter is scaled for a $\pm$ 10 volt maximum output, whereby the most significant bit in the y data being outputted corresponds to plus or minus 5 volts, depending on sign.

The parameters chosen for the sine/cosine cycling test are given in Table 10.

The resultant fundamental iteration rate is approximately 1.3×10^3 iterations per second and the sinusoidal angular frequency is 2.6 rad/sec. The resultant sine/cosine amplitude, A, is 2^7 and the outputted voltage amplitude peak from the converter is 7.5 volts.

The parameters chosen for the sine/cosine rotation test are given in Table 11. Additionally, the two segments of the sinusoidal curve chosen to show the fine-grain structure start at 0 radians and 0.5 radians. The resultant angular rotation increments for the 1x, 4x and 10x sets of integrators are 2^{-8}, 2^{-10} and 2^{-11} radians respectively.

Table 10. Sine/cosine cycle test parameters

Integrator Number	Bit Length n	y Register Variable	y Initial Value	Comments
1-1x	15	k_{1x}	2^{14}	
2-1x	8	A cos (ωt)	0.75×2^{8}	
3-1x	8	-A sin (ωt)	0	
4-1x	8	A sin (ωt)	0	Output 1x
1-4x	15	k_{4x}	2^{14}	
2-4x	10	A cos (ωt)	0.75×2^{8}	
3-4x	10	-A sin (ωt)	0	
4-4x	8	A sin (ωt)	0	Output 4x
1-10x	15	k_{10x}	0.8×2^{14}	
2-10x	11	A cos (ωt)	0.75×2^{8}	
3-10x	11	-A sin (ωt)	0	
4-10x	8	A sin (ωt)	0	Output 10x

Table 11. Sine/cosine rotation test parameters

Integrator Number	Bit Length n	y Register Variable	y Initial Value
1-1x	15	k_{1x}	1
2-1x	8	A cos $(\omega t - \frac{\pi}{2})$	0
3-1x	8	-A sin $(\omega t - \frac{\pi}{2})$	0.75 x 2^8
4-1x	11	$k_{1x}t$	0
1-4x	15	k_{4x}	1
2-4x	10	A cos $(\omega t - \frac{\pi}{2})$	0
3-4x	10	-A sin $(\omega t - \frac{\pi}{2})$	0.75 x 2^8
4-4x	11	$k_{4x}t$	0
1-10x	15	k_{10x}	1
2-10x	11	A cos $(\omega t - \frac{\pi}{2})$	0
3-10x	11	-A sin $(\omega t - \frac{\pi}{2})$	0.75 x 2^8
4-10x	11	$k_{10x}t$	0

V. SAMPLE PROBLEM EXPERIMENTAL DATA

The sample problem defined in section IV was programmed and run on the incremental processor.

The sine/cosine cycling test as shown by the mapping in Figure 18 was programmed and run using the parameters in Table 10. The resultant converter output was recorded using a Sanborn Model 150 SB-2 4-channel recorder. The resultant recorded output is shown in Figure 24.

The cycling test was performed by recording the first few cycles of the sinusoid, stopping the recorder for five minutes, and then continuing the recording. The resultant graph then indicates the sine generation over greater than 100 cycles of the function.

The expected amplitudes of 7.5 volts and angular frequency of 2.6 radians per second are indicated in Figure 24 and the gross structure essentially repeats over many cycles.

The sine/cosine rotation test as mapped in Figure 19 was programmed and run using the parameters of Table 11.

The incremental rotations were performed by repeatedly allowing computation to be performed until a time-scaler output increment is detected in integrator-4 y register; then stopping the computation to allow reading the y register contents of the cosine register.

The resultant data is plotted as shown in Figure 25 for the sigment starting at 0 radians and in Figure 26 for the

Figure 24. Since/cosine cycling test

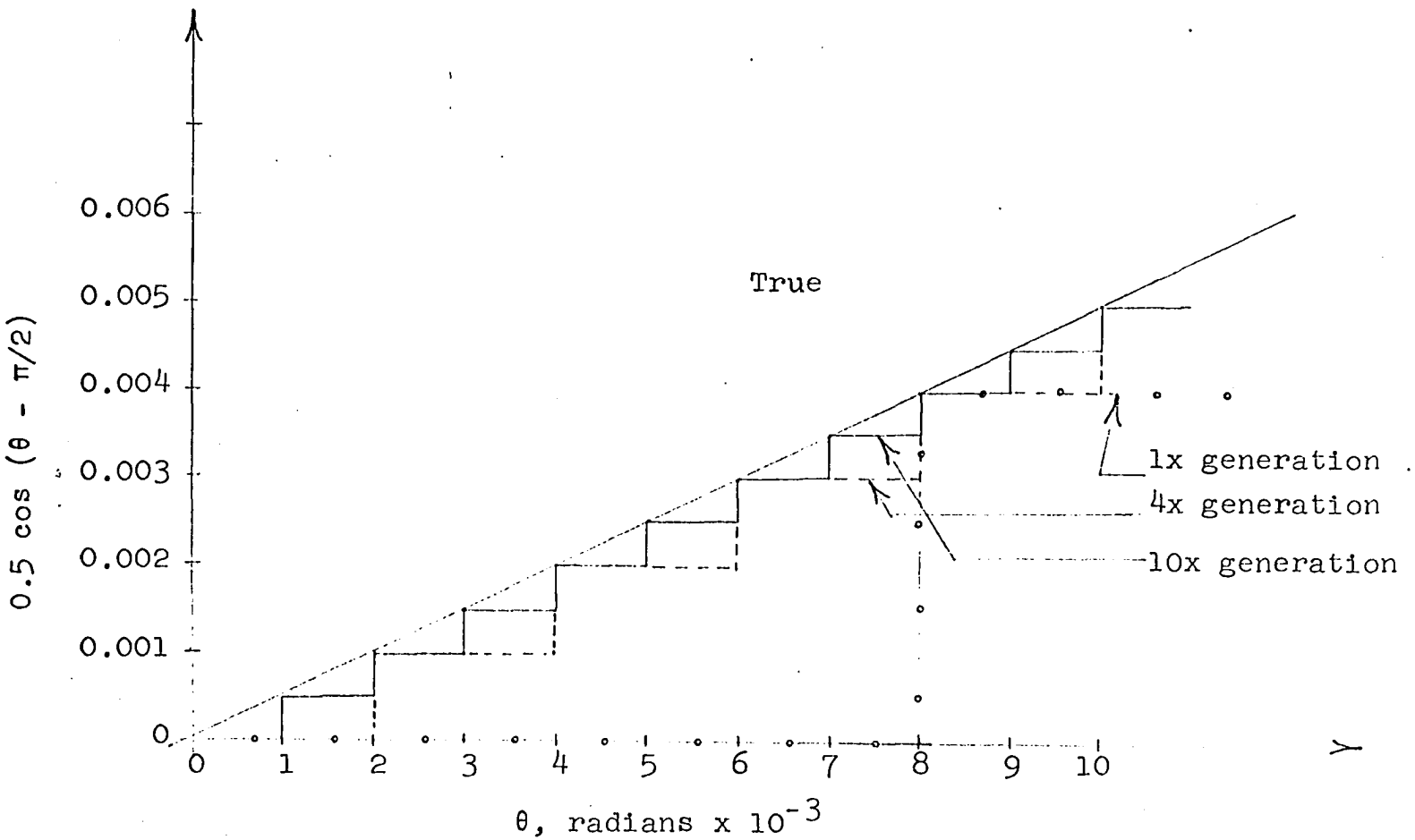


Figure 25. 0 radians segment incremental rotation

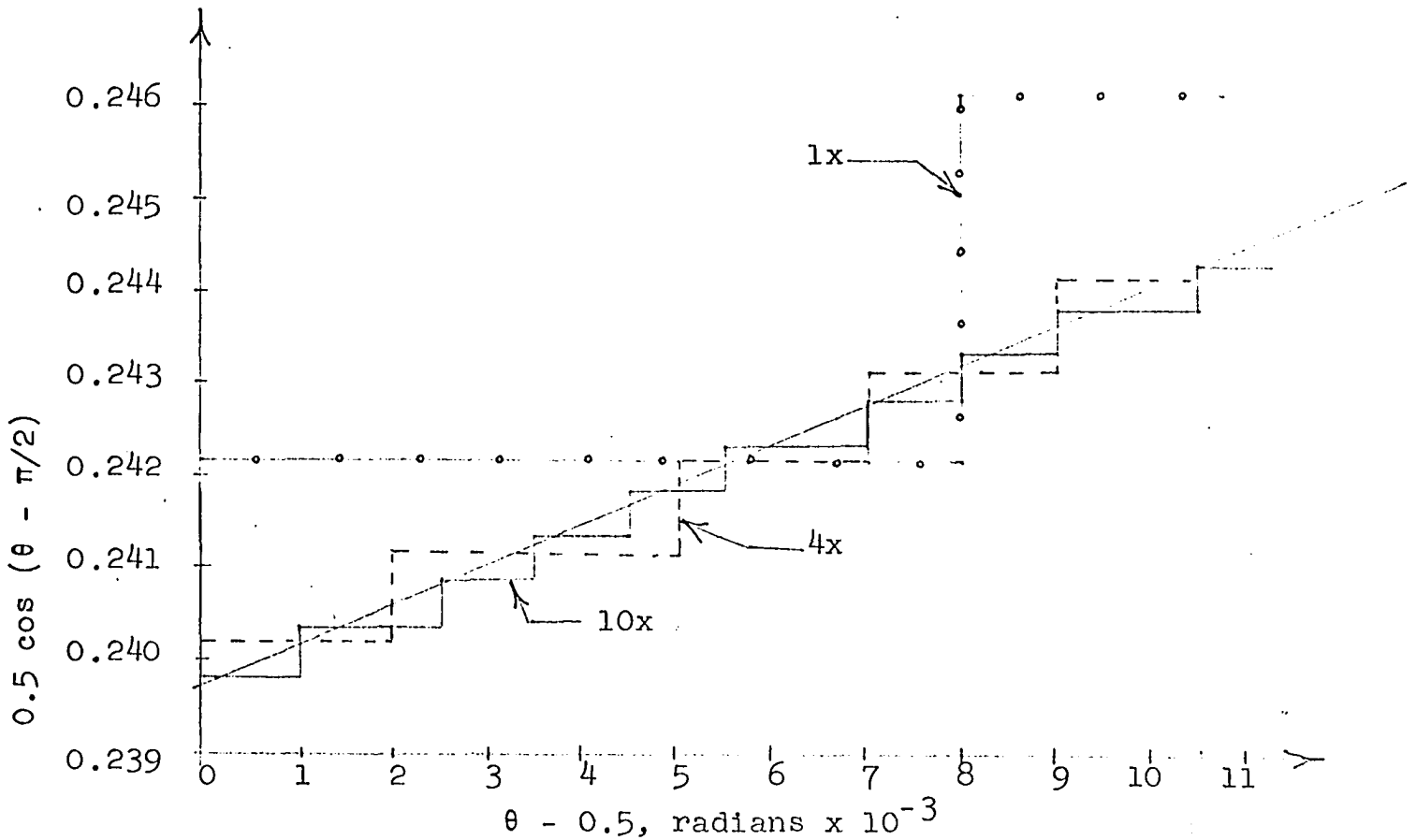


Figure 26. 0.5 radians segment incremental rotation

segment starting at 0.5 radians.

The improvement in sinusoidal generation is clearly indicated over both segments of the curve using the higher iteration rates.

The hardware used to implement the incremental processor consists for the most part of Collin's C-8400-series logic cards and rack hardware.

The front view of the processor is shown in Figure 26 which shows the control panel, the nine card rows holding 54 cards each, and the power-supply front panel.

The back view of the processor is shown in Figure 27. The point-to-point wiring between card plugs, the digital-to-analog converter near the top of the rack and the shielding case holding the memory plane located directly behind the front panel are illustrated in this back view.

Figure 28 shows the 128 word, 8 bit per word thin-magnetic-film plane, its shielding case and the associated diode-transformer matrix located next to the memory plane.

Two types of the Collins C-8400-series logic cards used in the incremental processor are shown in Figures 29 and 30. Figure 29 shows a KA-series logic inverter and Figure 30 shows an RS-series set-reset flip-flop.

The operating control panel for the equipment is shown in Figure 31. The row of indicators at the top of the panel are used to display selected data within memory and directly

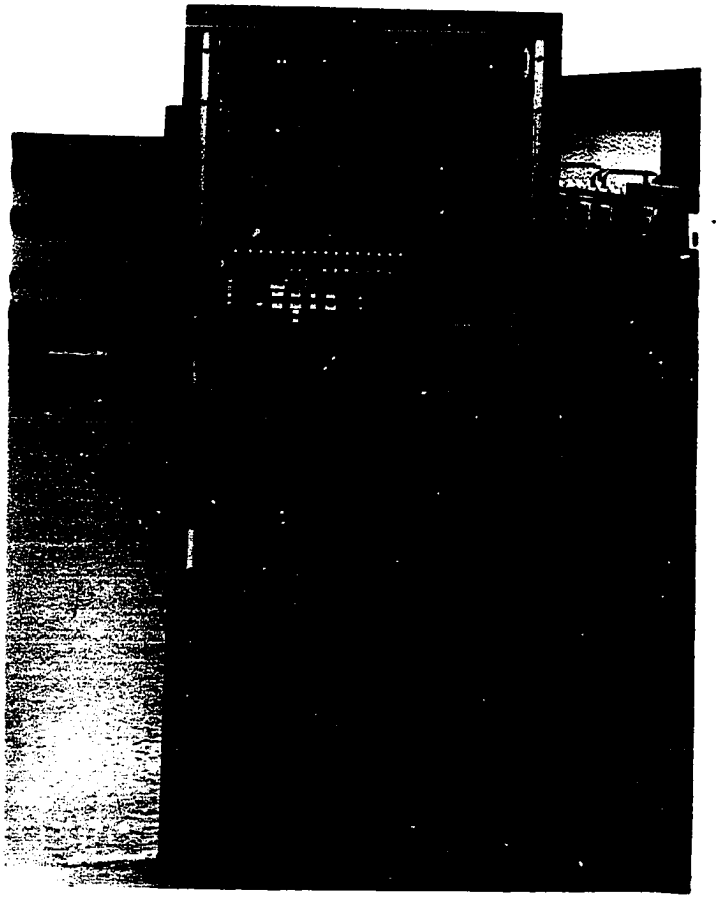


Figure 26. Processor front view

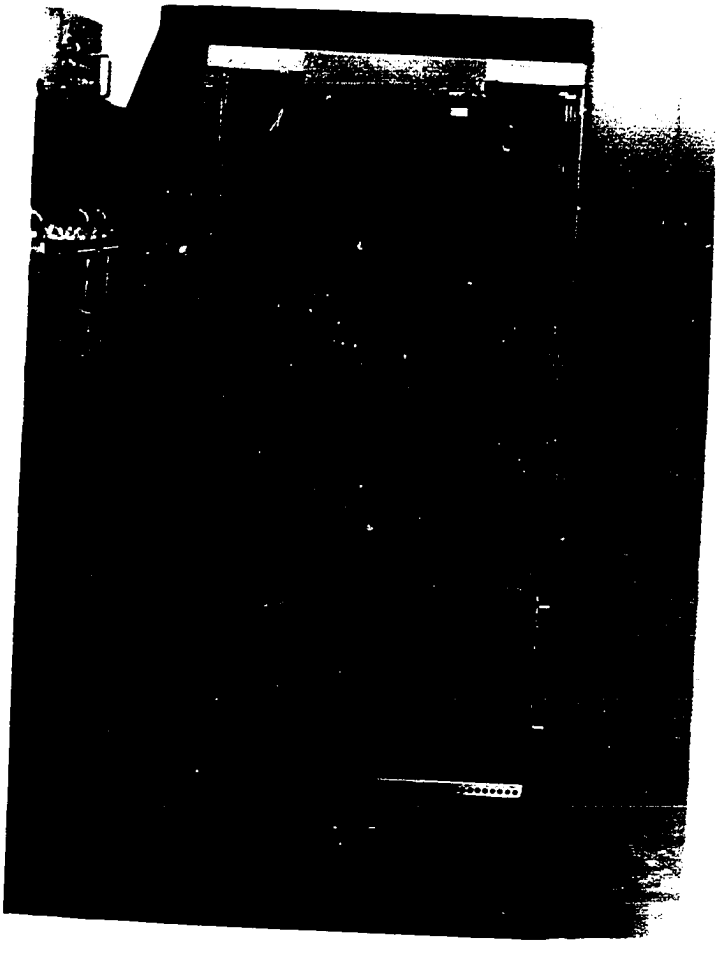


Figure 27. Processor back view

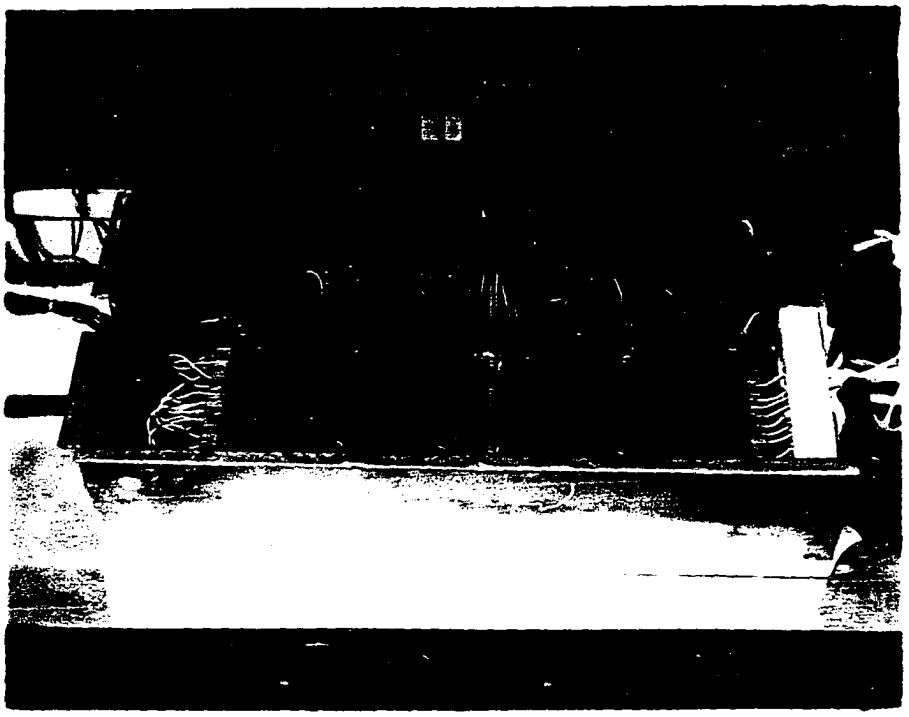


Figure 28. Thin-magnetic-film memory plane

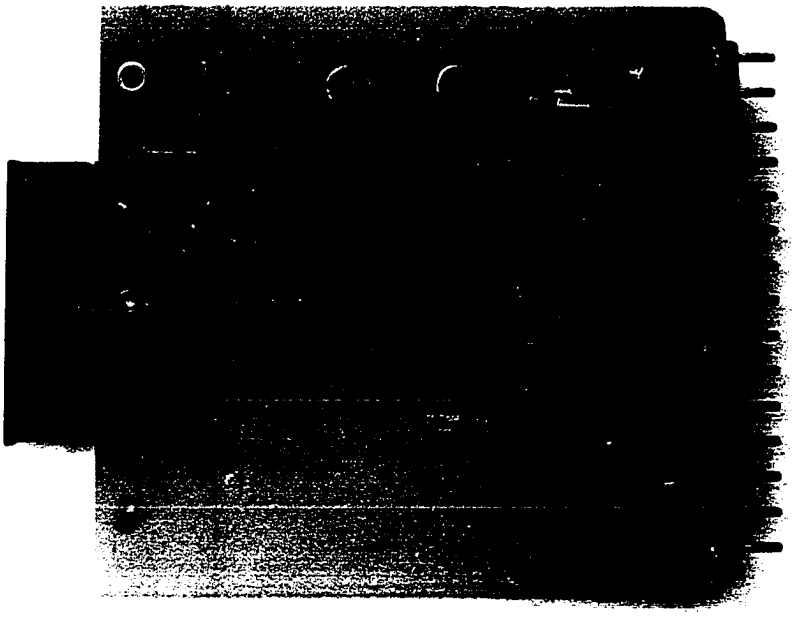


Figure 29. Collins KA-series logic inverter

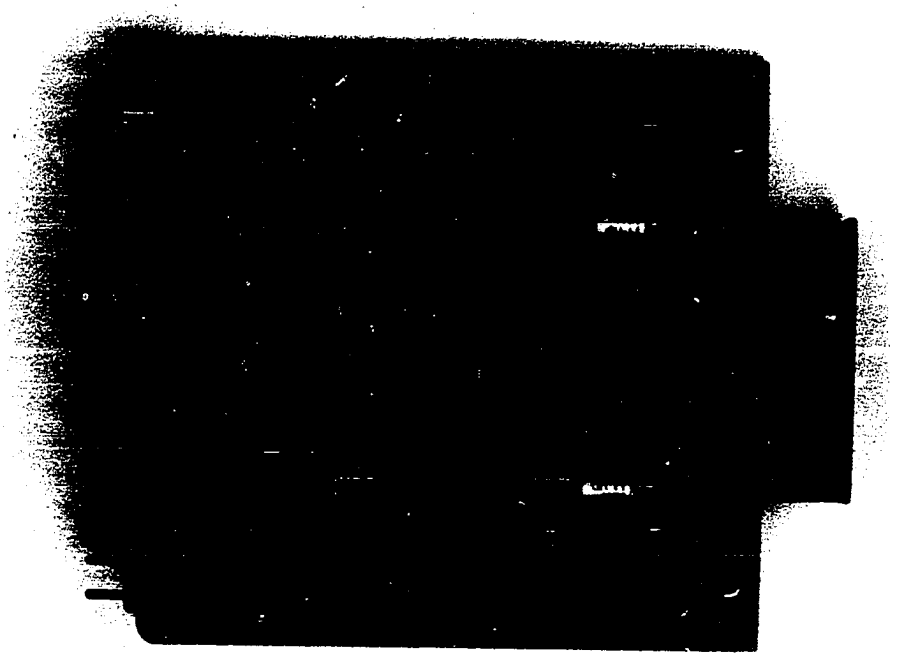


Figure 30. Collins RS-series set-reset flip-flop

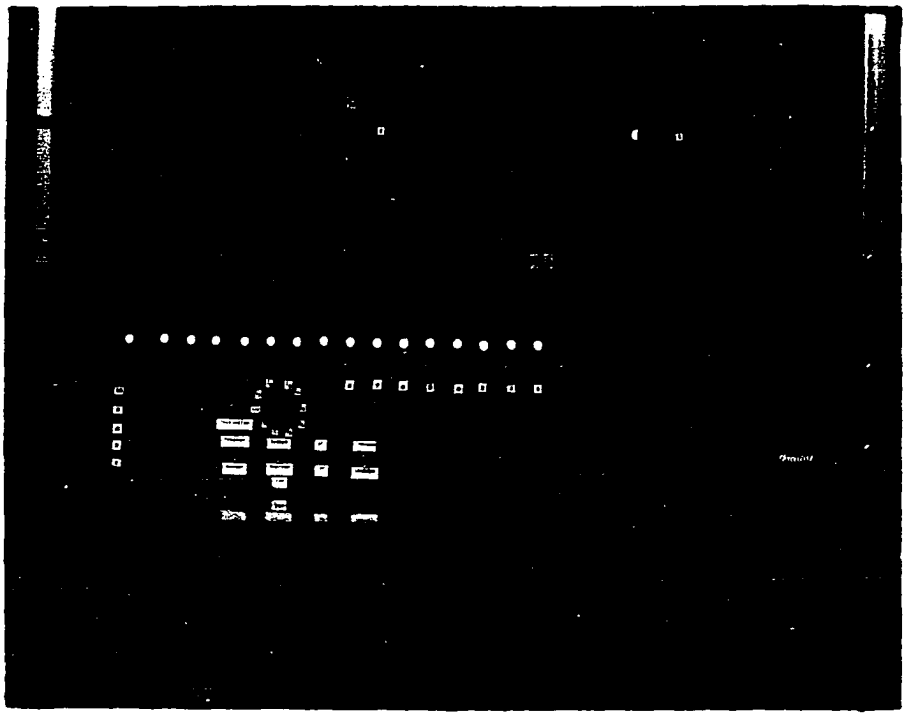


Figure 31. Incremental processor control panel

below these indicators are the push-button switches used to load data into memory. Below the indicators and insertion switches are the various data-selection, program-insertion, initialization and operating-control switches.

VI. CONCLUSIONS

The investigation of a multiple iteration rate incremental processor reported in the preceding pages was concerned with the critical characteristics pertinent to such a processor. Specific characteristics which were considered useful to extend the application of incremental processing were presented as the following: a basic DDA-integrator functional operation; an easily determined stored program; minimal storage in the form of a magnetic array and multiple-iteration-rate operation for sets of the integrators.

An incremental processor was developed, built and tested which incorporated the above characteristics. The results obtained in running test programs on the processor demonstrated the feasibility of design of such an equipment. Additionally, the results of the test programs illustrated the significant improvement in computation effected by use of multiple-iteration rates.

Several areas of application are presently being considered for use of such a processor. These include industrial process control, numerical-machine-tool control and data accumulation and reduction.

An area of research which warrants future investigation is the development of a useable technique for prediction of propagation of errors within an incremental processor. This error analysis has been discussed in the literature such as

in Monroe (16), Hills (17) and Nelson (18). A continuation of this effort to supply a complete and concise technique for application to a complete physical problem is needed.

VII. BIBLIOGRAPHY

1. Bush, V. and A. H. Caldwell. A new type of differential analyzer. Franklin Inst. J. 240: 255-326. 1945.

2. Korn, G. A. and T. M. Korn. Electronic analog computers. 2nd ed. New York, N.Y., McGraw-Hill Book Co., Inc. 1956.

3. Savant, C. F., Jr., R. C. Howard, C. B. Solloway and C. A. Savant. Principles of inertial navigation. New York, N.Y., McGraw-Hill Book Co., Inc. 1961.

4. Mendelson, J. J. The deimal digital analyzer. Aeronautical Engineering Review 13: 42-54. 1954.

5. Honeywell Military Products Group Aeronautical Division. The Honeywell 256-integrator digital differential analyzer. St. Petersburg, Fla., author. 1961.

6. Autonetics Division of North American Aviation, Inc. Marine-verdan computer technical description. Vol. 1. Downey, Calif., author. 1962.

7. Litton Industries. Technical description of Litton C-900 computer. Woodland Hills, Calif., author. 1960.

8. Mitchell, J. M. and S. Ruhman. The TRICE-a high speed incremental computer. Institute of Radio Engineers National Convention Record 6, Part 4: 206-216. 1958.

9. Bradley, R. E. and J. F. Genna. Design of a one-megacycle iteration rate DDA. Spring Joint Computer Conference Proc. 21: 353-364. 1962.

10. Computer Control Co., Inc. Manual of programming and operating instructions for the SPEC computer. Los Angeles, Calif., author. 1960.

11. Forbes, G. F. Digital differential analyzers. 4th ed. Los Angeles, Calif., Los Angeles Addressing and Mailing Co. 1957.

12. Bartee, T. C., I. L. Lebow and I. S. Reed. Theory and design of digital machines. New York, N.Y., McGraw-Hill Book Co., Inc. 1962.

13. Ledley, R. S. Digital computer and control engineering. New York, N.Y., McGraw-Hill Book Co., Inc. 1960.

14. Marcus, M. P. Switching circuits for engineers. Englewood Cliffs, N.J., Prentice-Hall, Inc. 1962.

15. Braun, E. L. Digital computer design. New York, N.Y., Academic Press, Inc. 1963.

16. Monroe, A. J. Digital processes for sampled data systems. New York, N.Y., John Wiley and Sons, Inc. 1962.

17. Hills, F. B. A study of incremental computation by difference equations: Report 7849-R-1. Cambridge, Mass., Servomechanisms Laboratory, Massachusetts Institute of Technology. 1958.

18. Nelson, D. J. DDA error analysis using sampled data techniques. Spring Joint Computer Conference Proc. 21: 365-375. 1962.

VIII. ACKNOWLEDGEMENTS

The author wishes to acknowledge the faculty of Iowa State University, both past and present, for providing the academic guidance which made this work possible. A special debt of gratitude is owed to Dr. R. G. Brown for his efforts, interest and encouragement.

The author is also grateful to Collins Radio Co., without whose support this work could not have been performed. Special thanks go to F. Matejcek and R. Schoon for their support in this work.

The author wishes to acknowledge the encouragement and understanding of his wife throughout these years of academic training and dissertation preparation.

IX. APPENDIX

The detailed logic circuitry used to implement the various logic functions within the incremental processor are described in the following paragraphs.

A preliminary discussion of the basic digital logic used is presented prior to specific function description.

The combinational logic equations synthesized within the equipment use the AND, OR and COMPLEMENT or NEGATE functions. These functions are used to implement the logic-select gates, the logic-decode gates, the code-generation circuitry and the binary adders.

Set-reset flip-flops are used for the storage registers and these set-reset flip-flops are also used in conjunction with a two-phase clock to implement the sequential counters.

The Collins Radio Co. C-8400-series inverter cards are used to implement the combinational-logic equations. The logic functions performed within the inverter cards is defined by use of the diagram in Figure 32. A corresponding typical logic expression for the inverter as shown in Figure 32 is given in Equation 42 where X is the output and A, B, C, D, E and F are the inputs.

$$X = \overline{(A \cdot B \cdot C + D \cdot E \cdot F)} \tag{42}$$

These inverter cards consist of the KA-series cards which incorporate one logic function as shown in Figure 32 per card and the KB-series cards which incorporate two independent

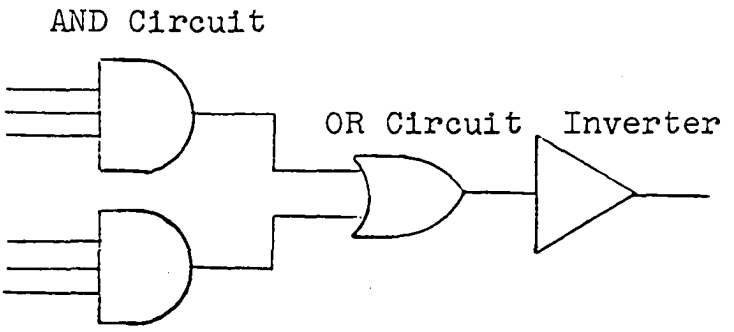


Figure 32. Collins inverter logic function

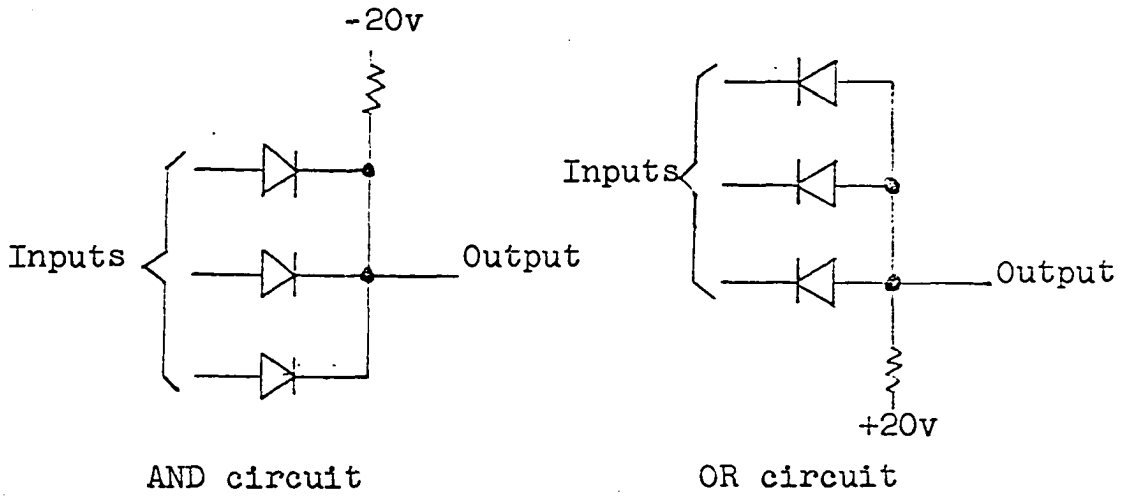


Figure 33. AND and OR diode-resistor circuits

logic functions as shown in Figure 32 per card.

The AND and OR functions are performed within the inverter cards by diode-resistor circuits as shown in Figure 33 and the inverter circuit consists of a Darlington-pair inverting switch circuit.

The set-reset flip-flops consist of the Collins RS-series cards which use the same circuitry as the KB-inverter cards with an additional internal cross connection between the two inverter functions as indicated in Equations 43 and 44.

$$Y = \overline{(A + \bar{Y})} \tag{43}$$

$$\bar{Y} = \overline{(B + Y)} \tag{44}$$

Internal cross connection

The set input for the flip-flop as implemented in Equations 43 and 44 is the logic variable B. The reset input is A.

The counting function is performed by use of two RS flip-flops for each counter stage and by use of a controlling two-phase clock.

The two-phase clock consists of the basic processor clock (denoted CLA) and a second clock signal (denoted CLB) which is the complement of CLA; i.e., CLA and CLB are alternately in logic state 1 and in logic state 0 such that CLA is in logic state 1 while CLB is in logic state 0 and vice versa.

The first of the two RS flip-flops for each stage (denoted XA) is utilized to store the present counter-stage state and the state of this flip-flop is changed when an input

is present. The second RS flip-flop (denoted XB) is used to store the past state of XA. The logic equations to implement these functions for the nth stage then are given in Equations 45 through 48 where I_n denotes the nth stage input.

$$\left.\begin{aligned} XA_n &= \overline{(I_n \cdot CLA \cdot XB_n + \overline{XA}_n)} \\ \overline{XA}_n &= \overline{(I_n \cdot CLA \cdot \overline{XB}_n + XA_n)} \end{aligned}\right\} \text{RS flip-flop} \qquad (45), (46)$$

$$\left.\begin{aligned} XB_n &= \overline{(CLB \cdot \overline{XA}_n + \overline{XB}_n)} \\ \overline{XB}_n &= \overline{(CLB \cdot XA_n + XB_n)} \end{aligned}\right\} \text{RS flip-flop} \qquad (47), (48)$$

The remaining function necessary to complete the entire counting function is the determination of the presence of an input to a specific stage. For up-counting, an input exists when the next-lower-order stage exhibits a past state of logic 1 and a present state of logic 0. The proper input for up-counting, I_{n-up}, is then given in Equation 49.

$$I_{n-up} = \overline{XA}_{n-1} \cdot XB_{n-1} \qquad (49)$$

For down-counting, an input exists when the next-lower-order stage exhibits a past state of logic 0 and a present state of logic 1. The proper input for down-counting, I_{n-down}, is then given in Equation 50.

$$I_{n-down} = XA_{n-1} \cdot \overline{XB}_{n-1} \qquad (50)$$

Substitution of Equation 49 or Equation 50 into Equations 45 and 46 yield the complete logic functions for the nth stage of the counter computer.

Storage registers are implemented by use of a single RS flip-flop per stage. The data is inputted to each stage under control of a command variable C_i. The resultant flip-flop logic expression for the nth stage is given in Equations 51 and 52 where D_n is the data to be inserted into the nth stage.

$$\left.\begin{aligned} X_n &= \overline{(C_i \cdot \bar{D}_n + \bar{X}_n)} \qquad (51) \\ \bar{X}_n &= \overline{(C_i \cdot D_n + X_n)} \qquad (52) \end{aligned}\right\} \text{RS flip-flop}$$

The logic-select gates consist of gates which transfer specific data to the output under control of a select-command variable as indicated in Equation 53.

$$Z = V \cdot SV + W \cdot SW + X \cdot SX + Y \cdot SY \qquad (53)$$

The variables in Equation 53 are the output, Z; inputs V, W, X and Y; and the selection-control variables SV, SW, SX and SY. Equation 53 is physically implemented by inverters connected to generate Equations 54 and 55 or alternately 56.

$$\bar{Z} = \overline{(V \cdot SV + W \cdot SW + X \cdot SX + Y \cdot SY)} \qquad (54)$$

$$Z = \overline{(\bar{Z})} \qquad (55)$$

$$Z = \overline{(\bar{V} \cdot SV + \bar{W} \cdot SW + \bar{X} \cdot SX + \bar{Y} \cdot SY)} \qquad (56)$$

The logic-decode gates are used to provide a logic 1 output in presence of specific code combinations on the inputs to the decode gate. The logic expression is derived by inserting all of the terms in canonical form for the specific codes desired and then reducing the expression to a minimal

form which can be physically implemented with the logic inverters.

An illustration of this is given where a 4-bit code pattern derived from a counter is used to generate an output on the count of 4, 7, 10 and 13. This was used specifically in the processor to derive the multi-rate code output for 4x rate. For the counter output denoted as Y_1, Y_2, Y_3 and Y_4 with Y_1 the least significant bit, the terms required in the logic expression are given in Equation 57.

$$Z = \underset{\text{"4"}}{\bar{Y}_4 \cdot Y_3 \cdot \bar{Y}_2 \cdot \bar{Y}_1} + \underset{\text{"7"}}{\bar{Y}_4 \cdot Y_3 \cdot Y_2 \cdot Y_1} + \underset{\text{"10"}}{Y_4 \cdot \bar{Y}_3 \cdot Y_2 \cdot \bar{Y}_1} + \underset{\text{"13"}}{Y_4 \cdot Y_3 \cdot \bar{Y}_2 \cdot Y_1} \quad (57)$$

The expression in Equation 57 can be reduced to the desired form as given in Equation 58.

$$Z = \overline{(\bar{Y}_3 \cdot \bar{Y}_4 + Y_1 \cdot \bar{Y}_3 \cdot + \bar{Y}_1 \cdot \bar{Y}_2 \cdot Y_4 + Y_2 \cdot Y_3 \cdot Y_4 + \bar{Y}_1 \cdot Y_2 \cdot Y_3 + Y_1 \cdot \bar{Y}_2 \cdot \bar{Y}_4)} \quad (58)$$

The code-generation circuitry is used to provide specific code combinations on parallel lines in the presence of specific inputs. The logic expression for the output on a specific bit line is derived in an analogous way to that for the logic-decode gates; i.e., the desired state of each bit line is established by the specific inputs required.

An illustration of this is given where a 3-bit code pattern is generated with outputs Z_1, Z_2 and Z_3 where the binary codes on the Z lines are to be binary 3 for an X input

and binary 6 for a Y input. The resultant three logic expressions are given in Equations 59, 60 and 61, where Z_1 is the least significant bit output.

$$Z_3 = \bar{X} + Y = \overline{(X \cdot \bar{Y})} \tag{59}$$

$$Z_2 = \bar{X} + Y = \overline{(\bar{X} \cdot \bar{Y})} \tag{60}$$

$$Z_1 = X + \bar{Y} = \overline{(\bar{X} \cdot Y)} \tag{61}$$

"3" "6"

The binary adder is implemented by use of inverters to generate the sum, S_n, and carry, C_n, for each nth parallel bit. The nth bit sum and carry logic expressions for the addition of X and Y are given in Equations 62 and 63 respectively.

$$S_n = X_n \cdot \bar{Y}_n \cdot \bar{C}_{n-1} + \bar{X}_n \cdot Y_n \cdot \bar{C}_{n-1} + \bar{X}_n \cdot \bar{Y}_n \cdot C_{n-1} + X_n \cdot Y_n \cdot C_{n-1} \tag{62}$$

$$C_n = X_n \cdot Y_n \cdot \bar{C}_{n-1} + X_n \cdot \bar{Y}_n \cdot C_{n-1} + \bar{X}_n \cdot Y_n \cdot C_{n-1} + X_n \cdot Y_n \cdot C_{n-1} \tag{63}$$

The carry function can be reduced to simpler form and the resultant logic functions generated are given in Equations 64 through 67.

$$\bar{S}_n = \overline{(X_n \cdot \bar{Y}_n \cdot \bar{C}_{n-1} + \bar{X}_n \cdot Y_n \cdot \bar{C}_{n-1} + \bar{X}_n \cdot \bar{Y}_n \cdot C_{n-1} + X_n \cdot Y_n \cdot C_{n-1})} \tag{64}$$

$$S_n = \overline{(\bar{S}_n)} \tag{65}$$

$$\bar{C}_n = \overline{(X_n \cdot Y_n + X_n \cdot C_{n-1} + Y_n \cdot C_{n-1})} \tag{66}$$

$$C_n = \overline{(\bar{C}_n)} \tag{67}$$