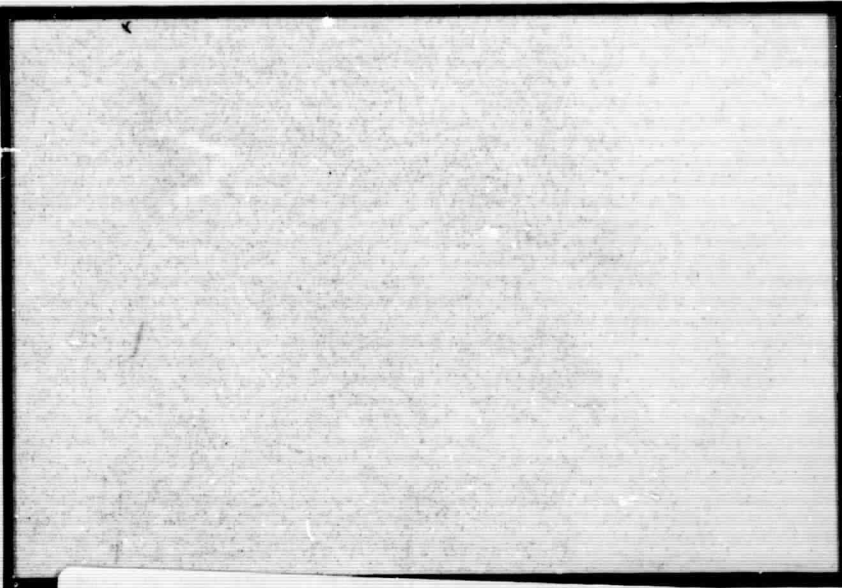


General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.
- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.
- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.
- This document is paginated as submitted by the original source.
- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

ELECTRICAL



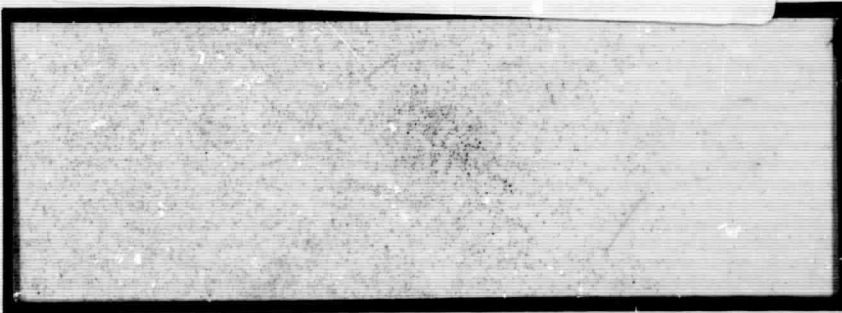
FACILITY FORM 602

N69-31293

(ACCESSION NUMBER) 99 (THRU) 1

(PAGES) NASA-CR 102307 (CODE) 08

(NASA CR OR TMX OR AD NUMBER) (CATEGORY)



ENGINEERING EXPERIMENT STATION

AUBURN UNIVERSITY

AUBURN, ALABAMA



REDUCING QUANTIZER DEADBAND WITH
A "RANGE SWITCHING" DIGITAL
FILTER

PREPARED BY

DIGITAL SYSTEMS LABORATORY

C. C. CARROLL, PROJECT LEADER

FIFTEENTH TECHNICAL REPORT

FEBRUARY, 1969

CONTRACT NAS8-11274
GEORGE C. MARSHALL SPACE FLIGHT CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
HUNTSVILLE, ALABAMA

APPROVED BY:

C. H. Holmes
C. H. Holmes
Head Professor of
Electrical Engineering

SUBMITTED BY:

C. C. Carroll
C. C. Carroll
Professor of
Electrical Engineering

FOREWORD

This report is a technical summary reporting the progress of a study conducted by the Digital Systems Laboratory of the Electrical Engineering Department, Auburn University, under the auspices of the Engineering Experiment Station. The study is focused toward fulfillment of Contract No. NAS8-11274, granted to Engineering Experiment Station, Auburn, Alabama, by the George C. Marshall Space Flight Center, National Aeronautics and Space Administration, Huntsville, Alabama.

REDUCING QUANTIZER DEADBAND WITH
A "RANGE SWITCHING" DIGITAL
FILTER

C. C. Carroll and J. R. Heath

ABSTRACT

This report is concerned with the design, implementation, and testing of a real-time digital filter in a suitable form for LSI realization. The design presented here was necessitated by the requirement that the deadband of the analog-to-digital converter be as small as possible. To accomplish this, a "range switching" filter was designed which employs shorter word lengths and consequently the physical size (component-wise) of the filter is reduced.

The compensator, as a special-purpose computer, is then tested in a simulation of the Saturn V thrust vector control system and the Pendulous Integrating Gyroscopic Accelerometer control loop for which it was designed.

ACKNOWLEDGEMENT

The authors wish to express appreciation to Mr. H. T. Nagle, Jr. for his valuable technical assistance and to Mr. M. O. Bedwell who compiled the figures and tables of this report.

TABLE OF CONTENTS

LIST OF FIGURES	vi
LIST OF TABLES.	viii
LIST OF SYMBOLS	ix
I. INTRODUCTION	1
II. MATHEMATICAL DERIVATION OF DIGITAL FILTER.	2
Direct Form	
Standard Form	
Modified Canonical Form	
Normal Form	
Selection of Programming Form	
"Range Switching" Process and Effects	
III. DIGITAL IMPLEMENTATION OF THE "RANGE SWITCHING"	
DIGITAL FILTER	23
Input-Output Equipment	
Arithmetic Unit	
Memory	
Control Unit	
IV. PERFORMANCE OF THE "RANGE SWITCHING" DIGITAL	
FILTER REALIZATION	70
Open-Loop Performance	
Closed-Loop Performance	
V. CONCLUSIONS.	86
REFERENCES.	88
APPENDICES.	90

LIST OF FIGURES

1.	Generalized block diagram of direct programming form.	5
2.	Generalized block diagram of standard programming form.	8
3.	Generalized block diagram of modified canonical programming form.	10
4.	Generalized block diagram for a $D(z)$ of $c/(z-\lambda)$	12
5.	Generalized block diagram of normal programming form.	12
6.	Generalized block diagram of second order digital compensator showing quantization points and internal variables.	15
7.	Decreasing quantization errors by paralleling two special purpose computers	17
8.	Decreasing quantization errors by switching digitally one special purpose computer.	17
9.	Block diagram of digital filter	24
10.	Generalized block diagram of digital filter showing the interrelation of the components of each group	25
11.	Logic diagram of analog-to-digital converter.	28
12.	Quantization characteristics of a successive approximation A/D round-off converter	29
13.	A bipolar-voltage D/A constructed from Honeywell μ -Pac logic modules	31
14.	Code translation logic for bipolar D/A.	32
15.	Input and output select logic	35

16.	Logic diagram of parallel binary accumulator.	40
17.	Shift registers of the Arithmetic Unit.	45
18.	Logic diagram of shift registers and accumulator combined to perform the multiplication process.	47
19.	Logic diagram of quantizers	50
20.	Output storage register	52
21.	Shift "up" or "down" logic of internal variable storage registers	54
22.	Saturation logic.	56
23.	Logic diagram of "range" memory and associated circuitry	57
24.	Coefficient storage	59
25.	Timing and pulse diagram for sequence of operation. . . .	62
26.	Control function generator.	63
27.	Data transfer logic for coefficients.	68
28.	Data transfer logic for internal variables.	69
29.	Open-loop step response	72
30.	Open-loop sinusoidal response	74
31.	Block diagram of PIGA control loop.	76
32.	Response of β to a unit step input.	77
33.	Response of β to a noise input.	78
34.	Response of "range switching" filter.	81
35.	Response of ideal case.	82
36.	Response with A/D quantization only	84
37.	Response of "non-range switching" ideal filter.	85

LIST OF TABLES

1.	Characteristics of programming forms.	6
2.	Code truth table	34

LIST OF SYMBOLS

A/D	analog-to-digital converter
β	mechanical angle to be nulled in the PIGA
D/A	digital-to-analog converter
D(z)	z-domain transfer function for a digital filter (or converter)
f-f	flip-flop
LSB	least significant bit
MSB	most significant bit
μ -Pac	commercially available digital logic modules from Honeywell, Inc.
n	order of the digital filter (or controller)
PIGA	pendulous integrating gyroscopic accelerometer
ϕ_d	attitude error angle including bending mode effects for the Saturn V t.v.c. system
SB	sign bit
S-P	special-purpose computer
T	sampling interval
t.v.c.	thrust vector control
z	e^{Ts} , the z-transform variable
z^{-1}	inverse z-transform of

I. INTRODUCTION

There have been numerous designs of digital filters, sometimes referred to as compensators or special purpose computers, proposed in recent literature [2,3,11,13,19]. All of these filters have the undesirable characteristic of a relatively wide quantization deadband. It is the purpose of the digital filter presented in this paper to reduce the quantization deadband and at the same time reduce the physical size of the filter. The design proposed here is a "range switching" digital filter which has predetermined quantization levels at three points within the filter.

The filter is employed to stabilize two control loops, the Pendulous Integrating Gyroscopic Accelerometer loop and the Saturn V thrust vector control system. The function of the filter in the two control loops mentioned is to physically realize a general second-order z-plane transfer function. Chapter II is devoted to the selection of a mathematical programming form which may be used to physically implement the z-plane transfer function.

After the mathematical programming form has been selected, each component of the filter must be digitally implemented using nand-gate logic and then performance tests run to determine the filters performance as is reported in Chapters III and IV respectively.

II. MATHEMATICAL DERIVATION OF DIGITAL FILTER

Before one can physically realize a z-plane transfer function, a mathematical programming form must be selected. There are many types of mathematical programming forms that could be used, but only four forms will be considered here. The four forms to be considered are:

- 1) Direct form,
- 2) Standard form,
- 3) Modified canonical form, and
- 4) Normal form.

From the above four forms, one will be chosen to physically realize the desired z-plane transfer function.

To determine the form to be used, several characteristics of the forms will be considered. They are:

- 1) the number of multiplications of feedback words by coefficient words,
- 2) the number of multiplications of input words by coefficient words,
- 3) the number of storage elements for input words,
- 4) the number of storage elements for feedback words,
- 5) the number of summing junctions, and
- 6) the number of compensator coefficient words.

All four forms will be compared using the above characteristics.

The form chosen will be the one which has a combination of the least number of the above characteristics. The reasons for considering these characteristics are given below.

First, the number of summing junctions determines the number of different sequences of calculations necessary during each sampling interval, and the complexity of the control circuitry is related directly to the number of sequences. Therefore, it is desirable to reduce the number of summing junctions.

Second, the number of storage elements should be kept small so as to reduce the physical component requirements of the computer as much as possible.

Third, the number of multiplications should be minimized so as to reduce the calculation time of the computer. This enables the computer to operate over a wider range of sampling frequencies.

Fourth, the computer should be constructed so that it is easily adaptable to any $D(z)$ of the form

$$D(z) = \frac{E_0(z)}{E_1(z)} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_n z^{-n}}{1 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_n z^{-n}} \quad (2-1)$$

which is the general form of a z-plane transfer function.

The characteristics previously listed as items (1) through (6) will now be examined for each of the four programming forms.

Direct Form

To construct the direct mathematical programming form, the generalized z-plane transfer function

$$D(z) = \frac{E_o(z)}{E_i(z)} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_n z^{-n}}{1 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_n z^{-n}} \quad (2-2)$$

can be expressed in the form

$$\begin{aligned} E_o(z) + b_1 z^{-1} E_o(z) + b_2 z^{-2} E_o(z) + \dots + b_n z^{-n} E_o(z) \\ = a_0 E_i(z) + a_1 z^{-1} E_i(z) + a_2 z^{-2} E_i(z) + \dots + a_n z^{-n} E_i(z). \end{aligned} \quad (2-3)$$

Now (2-3) is transformed to the time domain.

$$\begin{aligned} E_o(k) = a_0 E_i(k) + a_1 E_i(k-1) + a_2 E_i(k-2) + \dots + a_n E_i(k-n) \\ - b_1 E_o(k-1) - b_2 E_o(k-2) - \dots - b_n E_o(k-n) \end{aligned} \quad (2-4)$$

Equation (2-4) is the difference equation of a general z-plane transfer function expressed in direct programming form. A generalized block diagram for this form is shown in Fig. 1. The important characteristics that have to be considered are shown in Table 1. It is readily seen from Fig. 1 that this form requires $2n$ delay elements and only one junction at which summation and multiplication takes place.

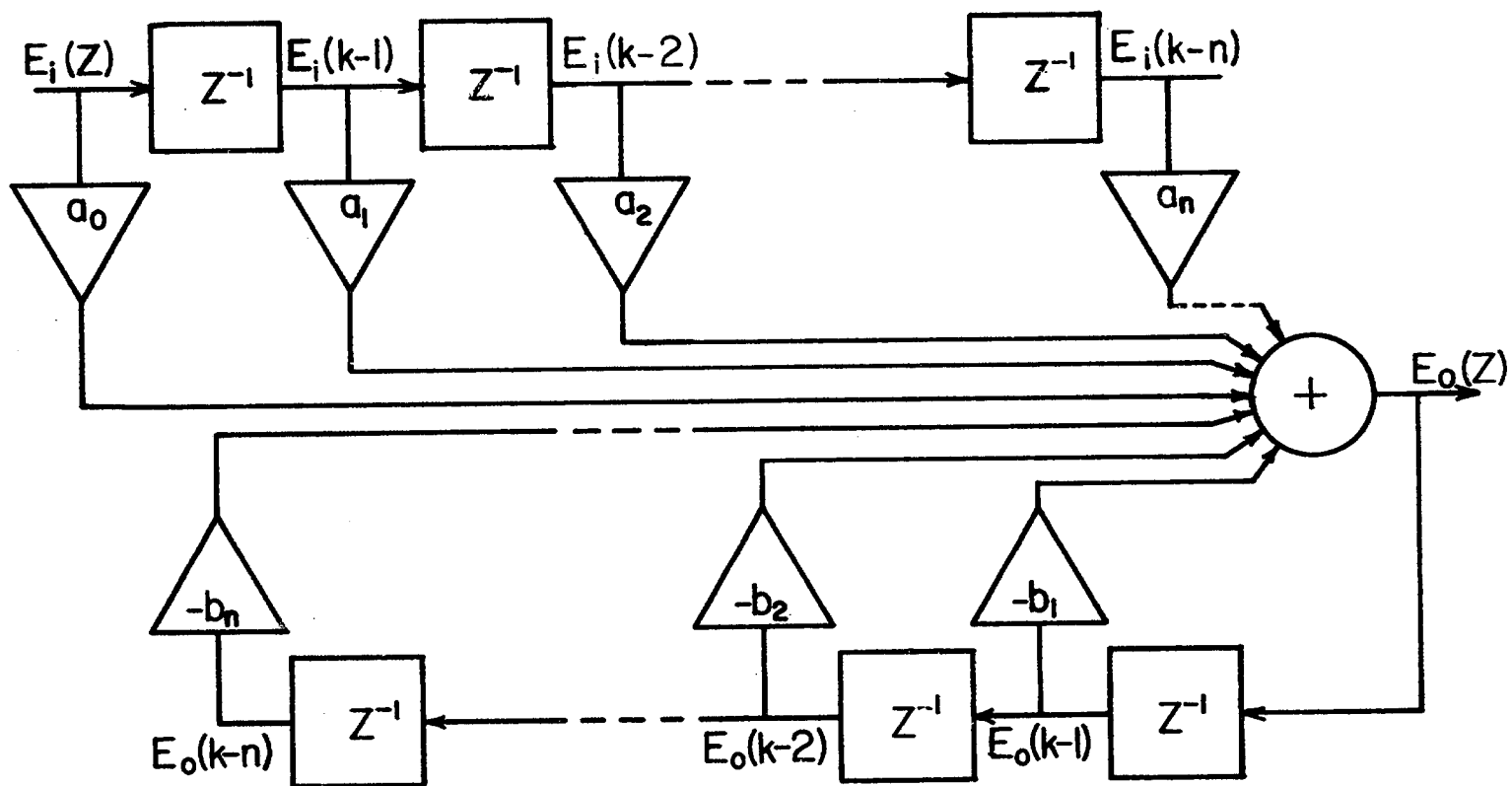


Fig. 1. Generalized block diagram of direct programming form.

Table 1. Characteristics of programming forms.

Item	Number Required For Each Programming Form			
	Direct	Standard	Canonical	Normal
1. Feedback coefficient mult.	n	n	$2n$	$2n$
2. Input coefficient mult.	$n+1$	$n+1$	1	1
3. Input word storage ele.	n	0	0	0
4. Feedback word storage ele.	n	n	n	n
5. Summing junctions	1	$n+1$	2	$n+1$
6. Coefficient words	$2n+1$	$2n+1$	$2n+1$	$2n+1$
Sum	$6n+3$	$6n+3$	$5n+4$	$6n+3$

Standard Form

The difference equation (2-4) is used to get the standard mathematical programming form; the equation will be repeated here.

$$E_o(k) = a_o E_i(k) + a_1 E_i(k-1) + a_2 E_i(k-2) + \dots + a_n E_i(k-n) \\ - b_1 E_o(k-1) - b_2 E_o(k-2) - \dots - b_n E_o(k-n). \quad (2-5)$$

Equation (2-5) can be physically realized as shown by the generalized block diagram of Fig. 2. Note that in Fig. 2 there are no " a_n " coefficients; instead " α_n " coefficients are used. This is done to show that the " a_n " coefficients in equation (2-5) have to be calculated in terms of the α 's and b 's in Fig. 2. As an example, for a second order system, $a_o = \alpha_2$, $a_1 = \alpha_2 + \alpha_2 b_1$, and $a_2 = \alpha_o + \alpha_1 b_1 + \alpha_2 b_2$. This is found to be true if a signal flow graph is drawn for the second order system of Fig. 2, and then the transfer function found from this. The z -plane transfer function for the second order system is

$$\frac{E_o(z)}{E_i(z)} = \frac{\alpha_2 + (\alpha_1 + \alpha_2 b_1)z^{-1} + (\alpha_o + \alpha_1 b_1 + \alpha_2 b_2)z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}}. \quad (2-6)$$

From Table 1, it is seen that the standard programming form has an advantage in that there are zero input-word storage elements but a disadvantage of having $n + 1$ junctions at which summation and multiplication take place. A disadvantage not seen from Table 1, but discussed

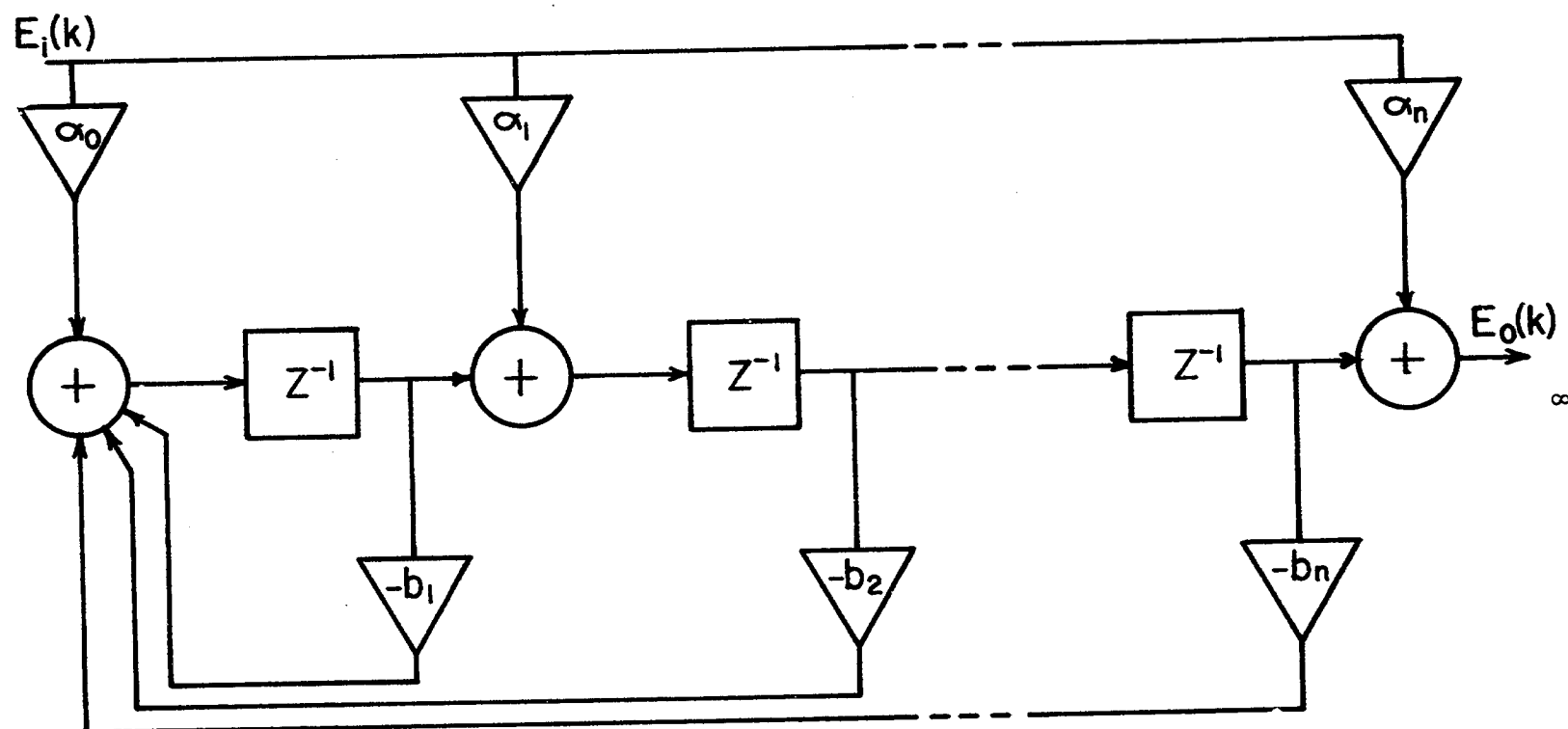


Fig. 2. Generalized block diagram of standard programming form.

earlier, is that the " α_n " coefficients of Fig. 2 have to be calculated from the " a_n " and " b_n " coefficients of equation (2-5). This is the main disadvantage of the standard programming form.

Modified Canonical Form

The next mathematical programming form to be discussed is the modified canonical form. To derive the difference equation of this form, an intermediate variable $M(z)$ is introduced into equation (2-2), which is the general z -plane transfer function. This intermediate variable is introduced as follows:

$$\frac{E_0(z)}{M(z)} = a_0 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_n z^{-n},$$

$$\frac{M(z)}{E_1(z)} = \frac{1}{1 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_n z^{-n}}. \quad (2-7)$$

The inverse z -transform of (2-7) is taken which results in the following difference equations,

$$E_0(k) = a_0 M(k) + a_1 M(k-1) + a_2 M(k-2) + \dots + a_n M(k-n) \quad (2-8)$$

and

$$M(k) = E_1(k) - b_1 M(k-1) - b_2 M(k-2) - \dots - b_n M(k-n). \quad (2-9)$$

Now equation (2-9) is substituted into equation (2-8), eliminating $M(k)$ from (2-8). After this is done (2-8) has the form

$$E_0(k) = a_0 E_1(k) + c_1 M(k-1) + c_2 M(k-2) + \dots + c_n M(k-n), \quad (2-10)$$

where $c_i = a_i - a_0 b_i$. Equations (2-9) and (2-10) are the two difference

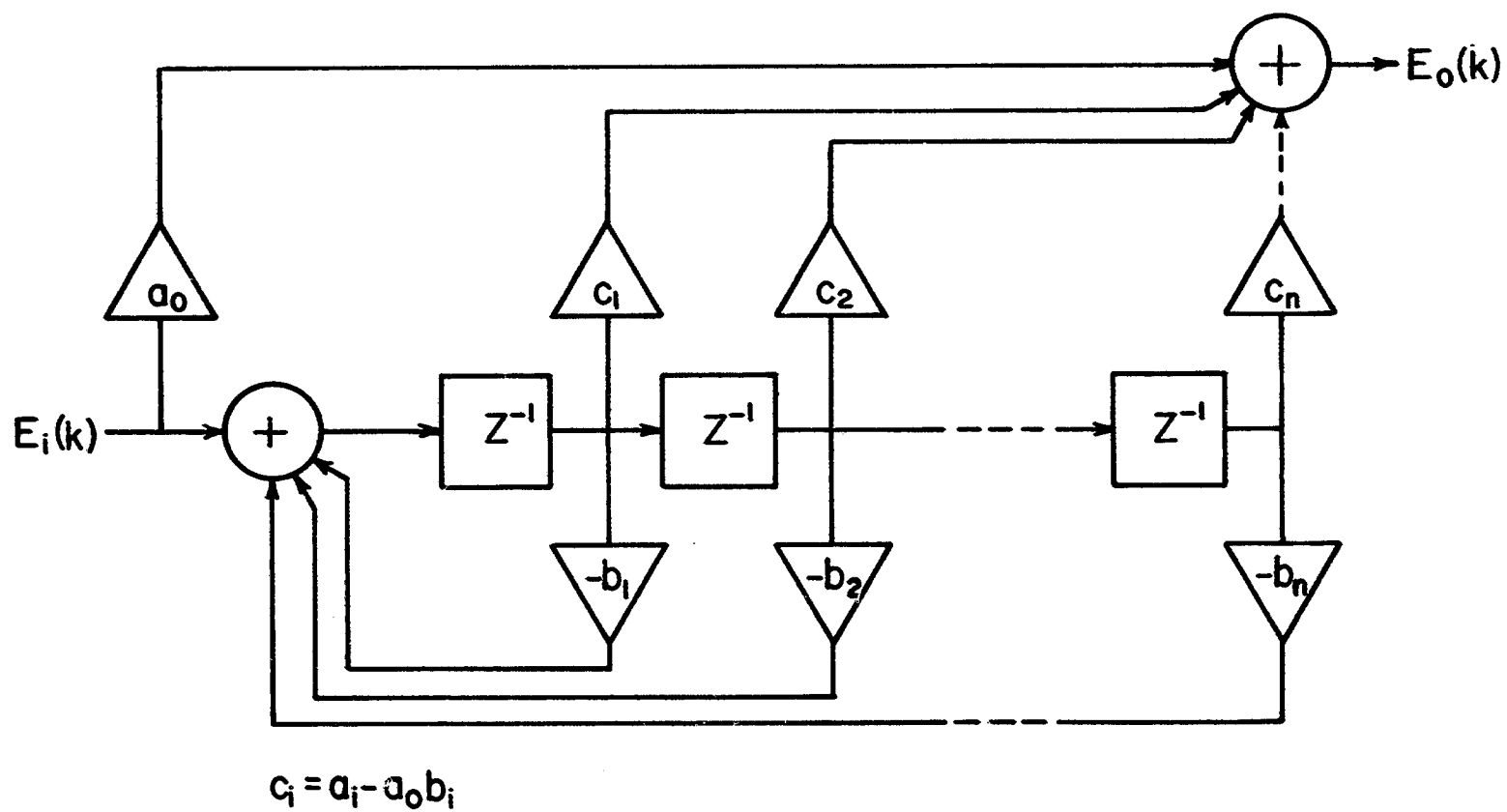


Fig. 3. Generalized block diagram of modified canonical programming form.

equations which will be mathematically programmed to get the canonical form. A block diagram of this form is shown in Fig. 3 and its associated characteristics in Table 1.

Examining the characteristics of the canonical form from Table 1, one sees that it has the advantage of zero input word storage elements, only one input coefficient multiplication and only two summation and multiplications junctions.

Normal Form

The last mathematical programming form that will be studied is the normal form. This form, like the three previous ones, is derived from the general z-plane transfer function which can be expanded as a sum of fractions as shown below,

$$\frac{E_o(z)}{E_i(z)} = \frac{c_1}{z-\lambda_1} + \frac{c_2}{z-\lambda_2} + \dots + \frac{c_n}{z-\lambda_n} . \quad (2-11)$$

Each term except the constant "d_o" on the right side of equation (2-11) can be physically realized by the generalized block diagram shown in Fig. 4. To physically realize equation (2-11), it is written in the form,

$$E_o(z) = E_i(z)d_o + E_i(z)\left[\frac{c_1}{z-\lambda_1}\right] + E_i(z)\left[\frac{c_2}{z-\lambda_2}\right] + \dots + E_i(z)\left[\frac{c_n}{z-\lambda_n}\right]. \quad (2-12)$$

A generalized block diagram of equation (2-12), which is the normal mathematical programming form is shown in Fig. 5.

Some of the more noticable characteristics of the normal programming form that are readily observable from Fig. 5 are that there

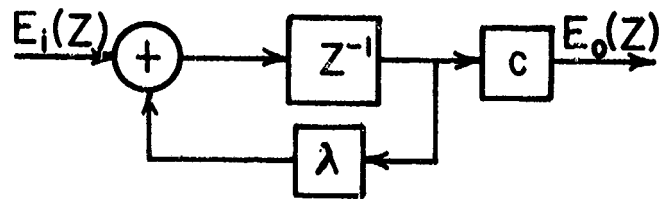


Fig. 4. Generalized block diagram
for a $D(Z)$ of $\frac{c}{Z-\lambda}$.

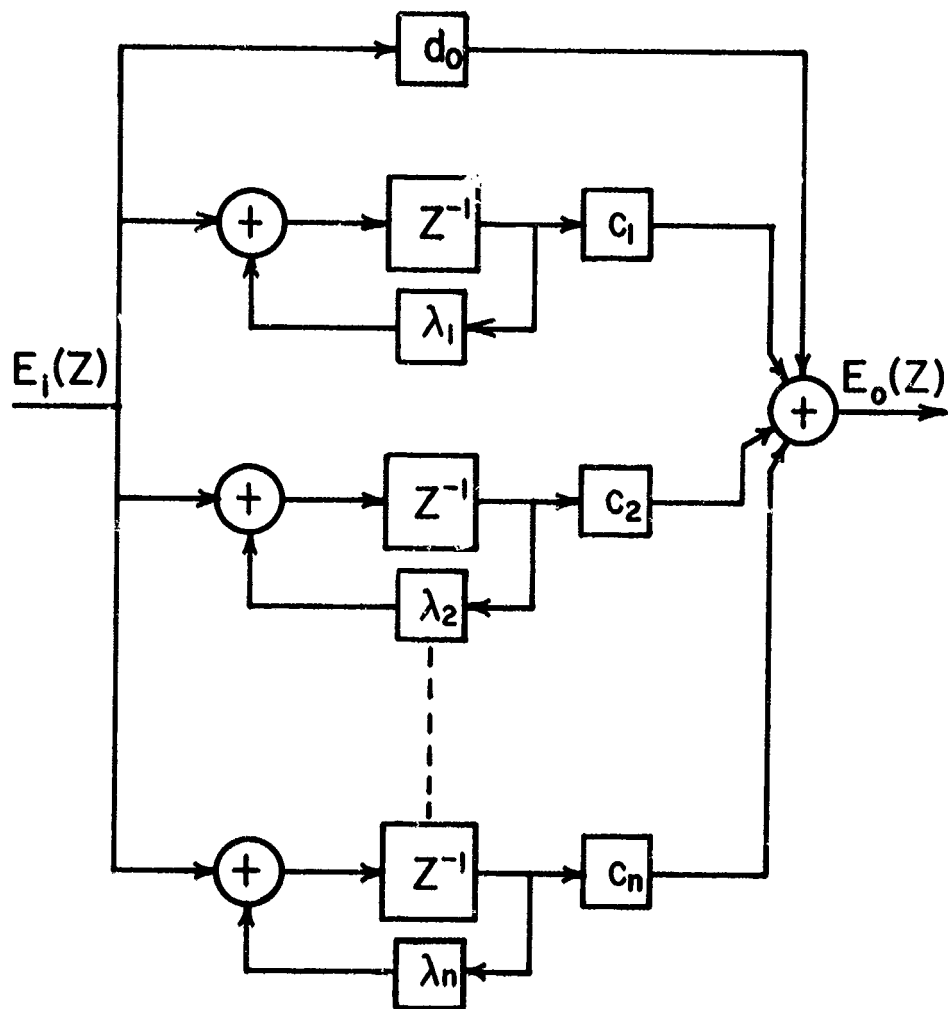


Fig. 5 Generalized block diagram
of normal programming form.

are $n + 1$ junctions at which summation and multiplication must take place and there are n delay elements. Its other characteristics are summarized in Table 1.

Selection of Programming Form

Now that the characteristics of the four mathematical programming forms have been found, the most suitable of these must be chosen for digital implementation. As stated earlier, the most desirable form is the one with a minimum number of summing junctions, storage elements, and required multiplications. It is also desirable to be able to adapt the form to any generalized, single input, single output, linear $D(z)$.

Observing Table 1, one can see that the canonical mathematical programming form is the most desirable. It has a combination of the least number of the six characteristics listed in Table 1. From the table, the direct form, the standard form, and the normal form each have a sum of $6n + 3$ characteristics, (n is the order of $D(z)$ being programmed) whereas the canonical form has a sum of $5n + 4$ which is less than $6n + 3$ for $n \geq 2$.

"Range Switching" Process and Effects

As explained in the introduction, a digital compensator (or controller) which will work both in the PIGA loop and the Saturn V t.v.c. system is being designed to physically realize the generalized second-order $D(z)$ as shown below

$$D(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \quad (2-13)$$

The compensator is to contain the feature of having quantization (a round-off process) at three points. The three points at which quantization takes place and the internal variables which will be used and explained later are shown in Fig. 6, which is a generalized block diagram of the modified canonical programming form of the second order compensator. The type of quantization and how it is accomplished at the three points in the compensator will now be explained.

The first point at which quantization takes place is at the input analog-to-digital (A/D) converter. For this particular compensator, an eight-bit signed-magnitude A/D converter is used; i.e., the continuous analog input signal is changed to an eight-bit signed-magnitude discrete digital signal suitable for the compensator. For the PIGA loop and the t.v.c. system, the eight-bit output of the A/D can have any integral magnitude between -255 and +255. The quantization of the A/D is accomplished by feeding only four of the eight bits into the compensator. The scheme used is to feed only the four most significant bits into the compensator if anyone of the four most significant bit positions is at logic "one," making the input of the compensator have a magnitude ranging between 16 and 255 in steps of 16. If this is done the four least significant bits are disregarded and not fed into the compensator. For the case when the four most significant bits are all of logic "0", only the four least significant bits are fed into the compensator; therefore in this case the compensator input will have a magnitude

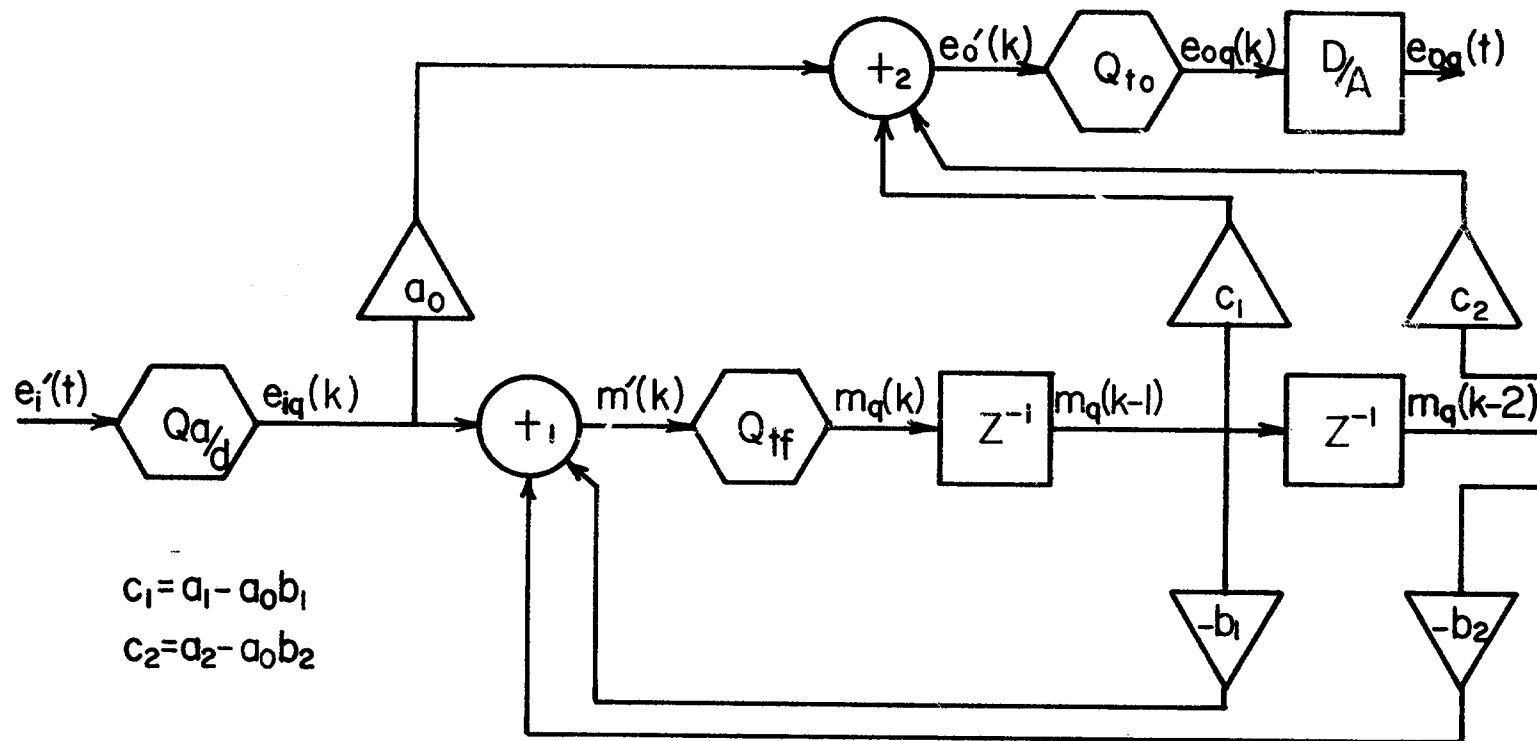


Fig. 6. Generalized block diagram of second order digital compensator showing quantization points and internal variables.

ranging between 1 and 16 in steps of one. This quantization process, in which only four bits are used by the compensator instead of all eight bits, has an advantage over feeding all eight bits into the compensator in that it reduces the digital circuitry in the compensator by a factor of one-third, thereby making the compensator smaller and simpler. It would seem that this quantization scheme would have an opposite effect of causing more circuitry to be required because of the two special-purpose (S-P) computers which might be needed, one to work on the four most significant bits when they are used and one to work on the four least significant bits when they are used as shown in Fig. 7. In Fig. 7, the output of the lower S-P computer must be multiplied by $1/16$ since the four bits fed into this computer have a magnitude of $1/16$ of those fed into the upper S-P computer.

So that only one S-P computer will be needed, a switching scheme is designed so that one computer works on both, the most significant bits when they are used and the least significant bits when they are used. A generalized block diagram of the compensator using the above scheme is shown in Fig. 8.

In explanation of how the digital switching compensator works during one sampling period, the analog input $e_1'(t)$ is first sampled by the A/D and an eight-bit digital approximation of $e_1'(t)$ appears at the output of the A/D. Next the input select logic selects four bits to be worked on by the special purpose computer. As stated earlier, if any of the four most significant bits are a logic "one," they are used by the S-P computer and if all four of the most significant

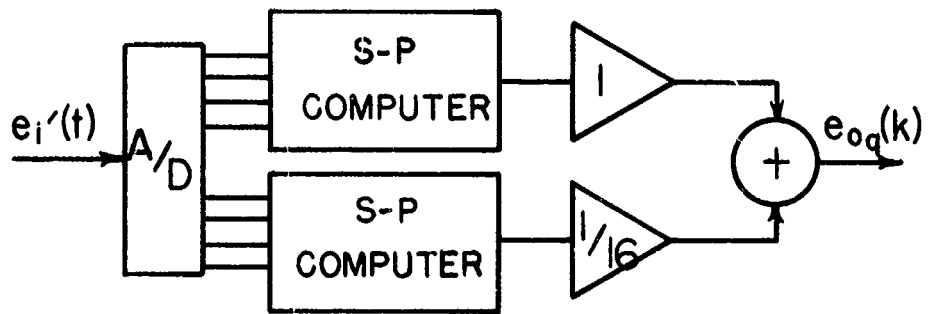


Fig. 7. Decreasing quantization errors by paralleling two special purpose computers.

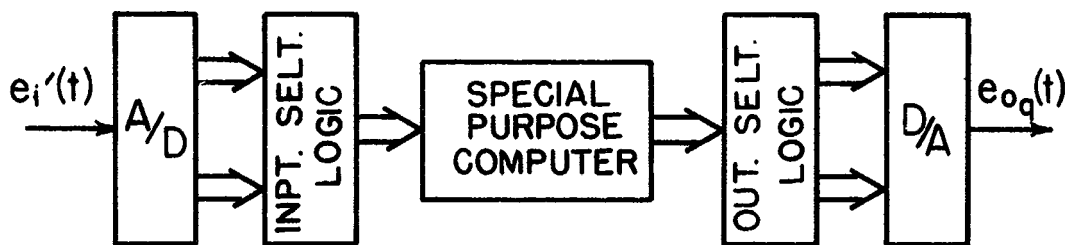


Fig. 8. Decreasing quantization errors by switching digitally one special purpose computer.

bits are at logic "0," the four least significant bits are inserted into the S-P computer. After the quantized digital input $e_{iq}(k)$ has been put into the S-P computer, the quantized output $e_o(k)$ is then calculated. Now the output select logic inserts the digital word $e_{oq}(k)$ into the D/A to be changed into an analog voltage level. The output select logic is needed so that the output digital word $e_{oq}(k)$ is properly weighted when it is changed from the digital word to the analog voltage level. When the four most significant bits have been used by the S-P computer, $e_{oq}(k)$ must be properly weighted such that it has a value sixteen times as great as it would if the four least significant bits had been used by the S-P computer.

From now on, to simplify the discussions, it is assumed that the compensator is using the four most significant bits when it is said to be working in the "higher range," and the four least significant bits when it is said that it is working in the "lower range."

Because of the range switching process just described, a method must be devised by which the two internal variables $m_q(k-1)$ and $m_q(k-2)$ of Fig. 6 are always properly weighted.

This is necessary because if the compensator were to remain in the same range each sampling period, the weight of $m_q(k-1)$ and $m_q(k-2)$ would not have to be changed, but if the compensator is working in the "lower range" and then changes to the "higher range" the weight of $m_q(k-1)$ and $m_q(k-2)$ has to be reduced by a factor of sixteen. This has to be done since the input range has been increased by a factor of sixteen. As an example of this, suppose that for a certain sampling period,

$m_q(k-1)$ has a magnitude of $(13)_{10}$ which is represented digitally as $(1101.00)_2$. Now suppose that on the next sampling period the input is sufficiently large such that the compensator will have to work in the "higher range." During this sampling period, $m_q(k-2)$ should have a magnitude of $(13)_{10}$, which was the magnitude of $m_q(k-1)$ the previous sampling period; however, since the compensator is working in the higher range, it sees $(1101.00)_2$ as having a magnitude of $(208)_{10}$, which is 16 times what it should be. Because of this, the binary point in $(1101.00)_2$ must be moved four places to the left to enable it to have the right weight.

For the case when the compensator changes from the "higher" to the "lower" range, the internal variables $m_q(k-1)$ and $m_q(k-2)$ have to be increased by a factor of sixteen. This is done by shifting the decimal point four places to the right. As an example of this, suppose that for a certain sampling period $m_q(k-1)$ has a value of $(2)_{10}$ which is represented digitally as $(000010.00)_2$ in the compensator. Now suppose that on the next sampling period, the compensator changes to the "lower range," or the input has a magnitude ranging between zero and fifteen. Since $m_q(k-1)$ is now $m_q(k-2)$, $m_q(k-2)$ should have a magnitude of $(2)_{10}$, but since the compensator is working in the lower range, it sees $(00010.00)_2$ as having a magnitude of $1/8$ which is $1/16$ of what it should be. To enable $m_q(k-2)$ to have a magnitude of $(2)_{10}$, the binary point must be moved four places to the right, thus $m_q(k-2)$ becomes $(0100000.00)_2$. It will be seen later that the $m_q(k-1)$ and $m_q(k-2)$ storage registers are not large enough to shift the binary

point four places to the right because logic "ones" in the four most significant bit positions are lost. Instead, a scheme is devised which shifts a logic "one" into all the bit positions if any one of the four most significant bit positions has a logic "one" in it, making the storage registers under these conditions have their maximum magnitude when the compensator shifts from the "higher" to the "lower" range.

It is seen from the above that when the compensator is working in the same range all the time, the $m_q(k-1)$ and the $m_q(k-2)$ storage registers always keep their same weight; that when the compensator changes from the "lower" to the "higher" range of operation, the magnitude of the $m_q(k-1)$ and the $m_q(k-2)$ storage registers is divided by a factor of sixteen and that when the compensator changes from the "higher" to the "lower" range, the values in the storage registers are multiplied by a factor of sixteen. How this process is accomplished digitally is explained in the next chapter.

The second place at which quantization takes place in the compensator is the output of the first summing junction within the compensator. The output of the first summing junction, $m'(k)$, is quantized to get the quantized value of the feedback variable $m(k)$. The quantization at this point is simply a round-off process. How this is digitally implemented will be explained in the next chapter. To determine how much $m'(k)$ could be quantized, the Fortran source programs of the simulation of the PIGA loop and the t.v.c. system shown in the appendices were used on the IBM 7040 computer. After trying many quantization levels, it

was decided that the best operation of the compensator was achieved when $m'(k)$ was quantized such that $m_q(k)$ could have a maximum value of (63.75). This was found to be true for both the FIGA and the t.v.c. systems. Digitally this meant that $m_q(k)$ could have a maximum of six bits to the left of the binary point and two bits to the right of the binary point.

To see how the quantization process takes place mathematically, $Q_{tf} [x]$ must first be defined. $Q_{tf} [x]$ means that the feedback quantization is acting on the feedback variable x . $Q_{tf} [x]$ is defined as

$$Q_{tf} [x] = sk \quad \text{where } k \leq |x|, s = \text{sign } x, k \text{ a real positive number}$$

$$Q_{tf} [x] = Q_{tf} [-x] .$$

From Fig. 6 and the definition of $Q_{tf} [x]$ it is seen that the following equations mathematically describe the quantization process at the output of the first summing junction.

$$\begin{aligned} m'(k) &= e_{iq}(k) - b_1 m_q(k-1) - b_2 m_q(k-2) \\ m_q(k) &= Q_{tf} [e_{iq}(k) - b_1 m_q(k-1) - b_2 m_q(k-2)] \end{aligned} \tag{2-14}$$

where $|Q_{tf} [e_{iq}(k) - b_1 m_q(k-1) - b_2 m_q(k-2)]| \leq 63.75$.

The last place at which quantization takes place in the S-P computer is at the output of the second summing junction as can be seen from Fig. 6. Here the discrete output variable $e_o'(k)$ is

quantized to get the discrete output variable $e_{oq}(k)$.

As before, to determine the amount of quantization needed at this point, the Fortran source program shown in the appendices which simulates the operation of both the PIGA and the t.v.c. system was used. It was found that the best operation of the systems was achieved when $e_o'(k)$ was quantized such that it had five bits to the left of the binary point and three bits to the right of the binary point. Therefore, $e_{oq}(k)$ can have a maximum magnitude of 31.875. The quantization process at this point is mathematically described by the equations,

$$e_o'(k) = a_o e_{iq}(k) + c_1 m_q(k-1) + c_2 m_q(k-2)$$

and

(2-16)

$$e_{oq}(k) = Q_{to} [a_o e_{iq}(k) + c_1 m(k-1) + c_2 m(k-2)],$$

where $|Q_{to} [a_o e_{iq}(k) + c_1 m(k-1) + c_2 m(k-2)]| \leq 31.875$.

III. DIGITAL IMPLEMENTATION OF THE "RANGE-SWITCHING" DIGITAL FILTER

Since the modified canonical programming form has been chosen, a means must be devised by which this form can be digitally implemented. It is the purpose of this chapter to discuss the approach taken to implement the filter and then describe how it is actually accomplished.

Observing the equations which are to be realized physically, (2-15) and (2-16), and Fig. 6, which is a generalized block diagram of the modified canonical programming form for the second order filter, one can see that its components can be organized into four groups. These four groups are 1) Input and Output Equipment, 2) Arithmetic Unit, 3) Memory and 4) the Controller. A generalized block diagram of the digital filter organized in this manner is shown in Fig. 9. The digital components contained in each group are described briefly below.

The Input-Output equipment consists of the analog-to-digital converter (A/D), the digital-to-analog converter (D/A), and the input and output select logic; the Arithmetic Unit consists of shift registers, a parallel binary accumulator, and the reduction logic; the Memory is composed of the coefficient storage, internal variable storage and the output storage; and lastly, the control function generator and the data transfer logic make up the Control Unit. A more detailed diagram of the digital filter which incorporates all the components of each group into the diagram is shown in Fig. 10. In Fig. 10,

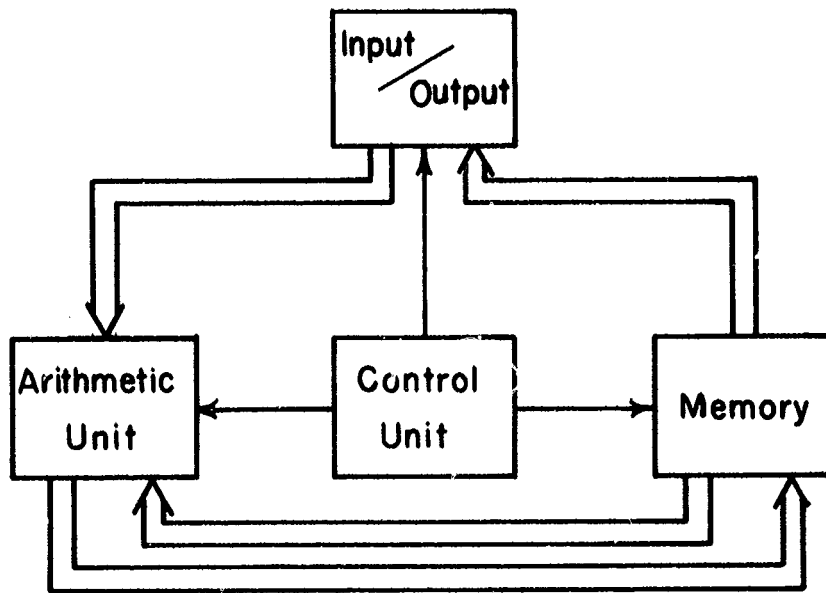


Fig. 9. Block diagram of digital filter.

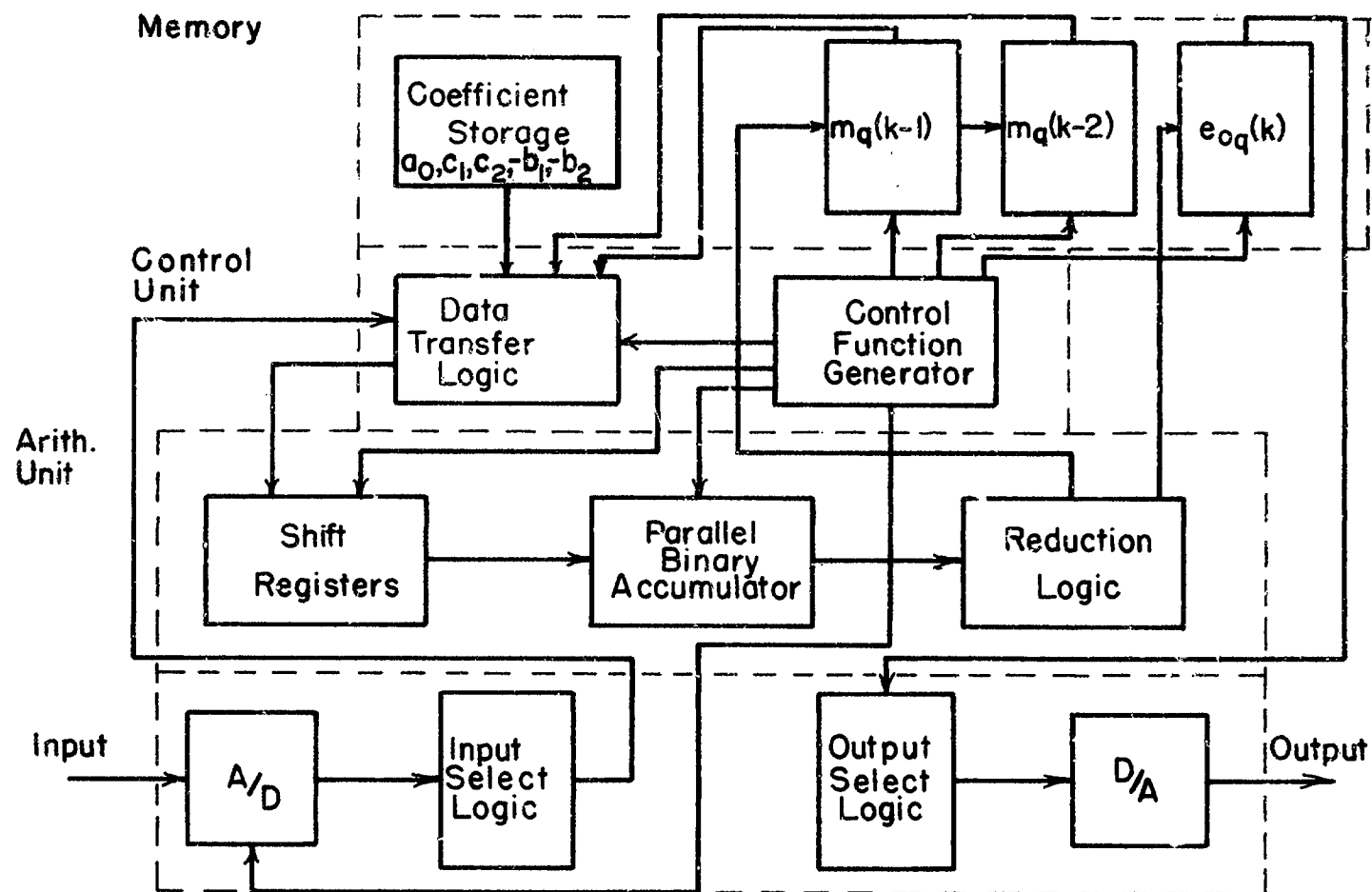


Fig. 10. Generalized block diagram of digital filter showing the interrelation of the components of each group.

the coefficients and the internal variables are labeled so that the relation between the figure and the two equations that have to be physically realized by the digital filter can be seen. The two equations that have to be physically realized by the second-order filter are shown again below

$$e_o'(k) = a_o e_{iq}(k) + c_1 m_q(k-1) + c_2 m_q(k-2) \quad (3-1)$$

$$m'(k) = e_{iq}(k) - b_1 m_q(k-1) - b_2 m_q(k-2) \quad (3-2)$$

where $c_i = a_i - a_o b_i$.

The remainder of this chapter is concerned with the construction of each component of Fig. 10 using Honeywell μ -Pac nand-gate logic. The components are discussed according to which group they are contained in.

Input-Output Equipment

It is the purpose of the input-output equipment to first convert the analog input signal of the digital filter into a signed-magnitude digital word so that the computer can operate on it and from this digital word, select the range that the computer will operate in during the present sampling period. Finally it converts a signed magnitude digital output word back into an analog signal which is the analog output of the filter. The way in which these three processes are accomplished is discussed below.

Analog-to-digital conversion.

As stated before, the purpose of the analog-to-digital converter (A/D) is to convert the analog input signal, $e_i(t)$ of Fig. 6, into the signed magnitude digital word $e_{iq}(t)$.

The A/D of Fig. 11 is a successive approximation converter which has the non-linear characteristics shown in Fig. 12. The A/D divides the input analog voltage range into 256 distinct magnitude levels which are separated by an amount h . The 256 distinct magnitude levels are represented by an eight-bit digital word, which is sufficient to meet the specifications required by the PIGA loop and the Saturn V t.v.c. system. It is seen from Fig. 12 that the quantizing error, $e_i(t) - e_i(k)$, is bounded by the expression

$$-\frac{h}{2} \leq e_i(t) - e_i(k) \leq \frac{h}{2}$$

This being true, the maximum magnitude of the input quantizing error is $h/2$.

The A/D that is to be used is constructed from the digital logic diagram of Fig. 11. The operation of the A/D converter is explained in full in [11]. Basically, the A/D converter receives the analog input signal, $e_i(t)$, and after eight approximation pulses are received by the converter, the eight bit digital word, $e_i(k)$, appears at its output. The control pulses required for operation of the A/D is a system reset pulse and eight A/D approximation pulses, one pulse for each bit in $e_i(k)$.

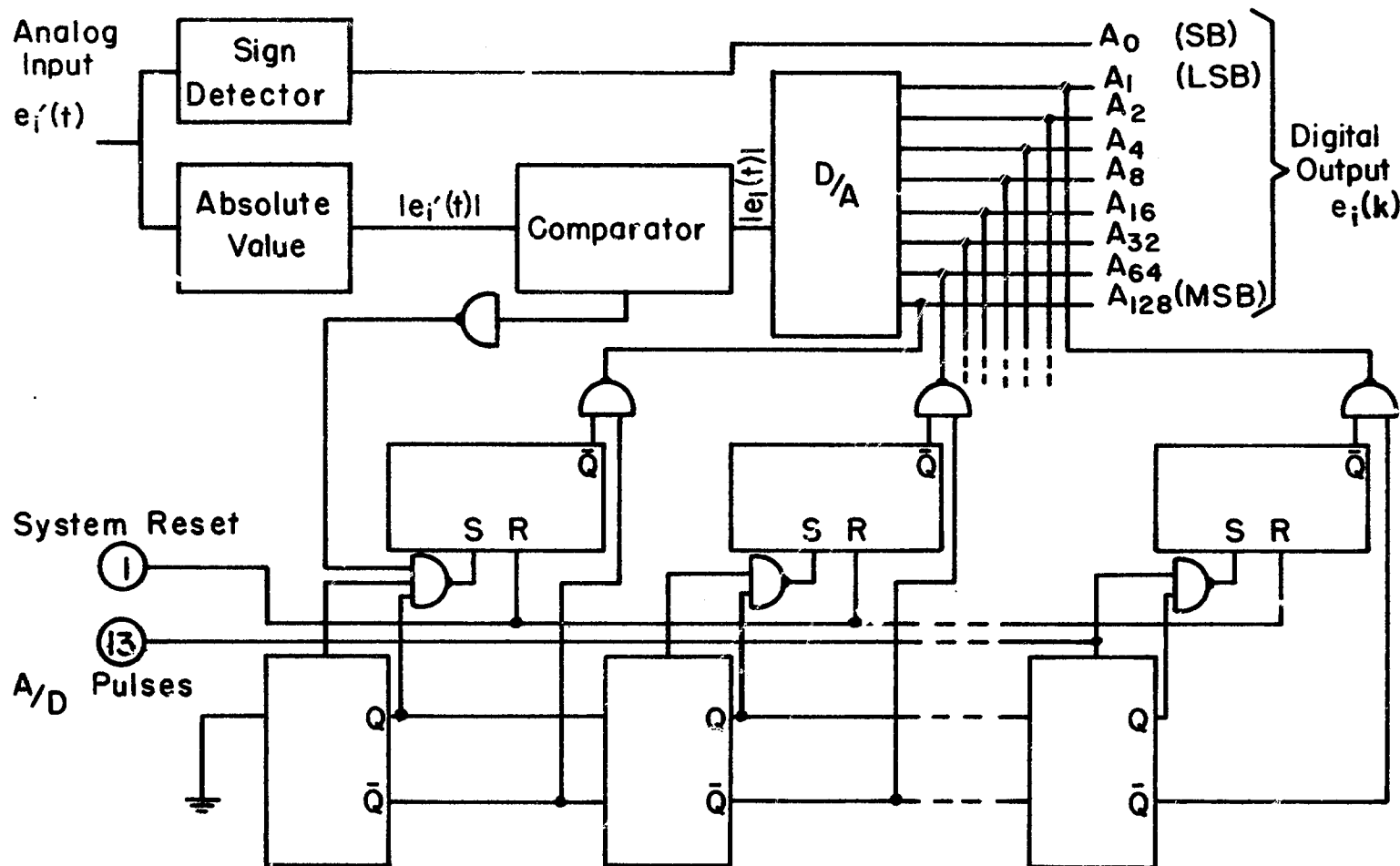


Fig. 11. Logic diagram of analog-to-digital converter.

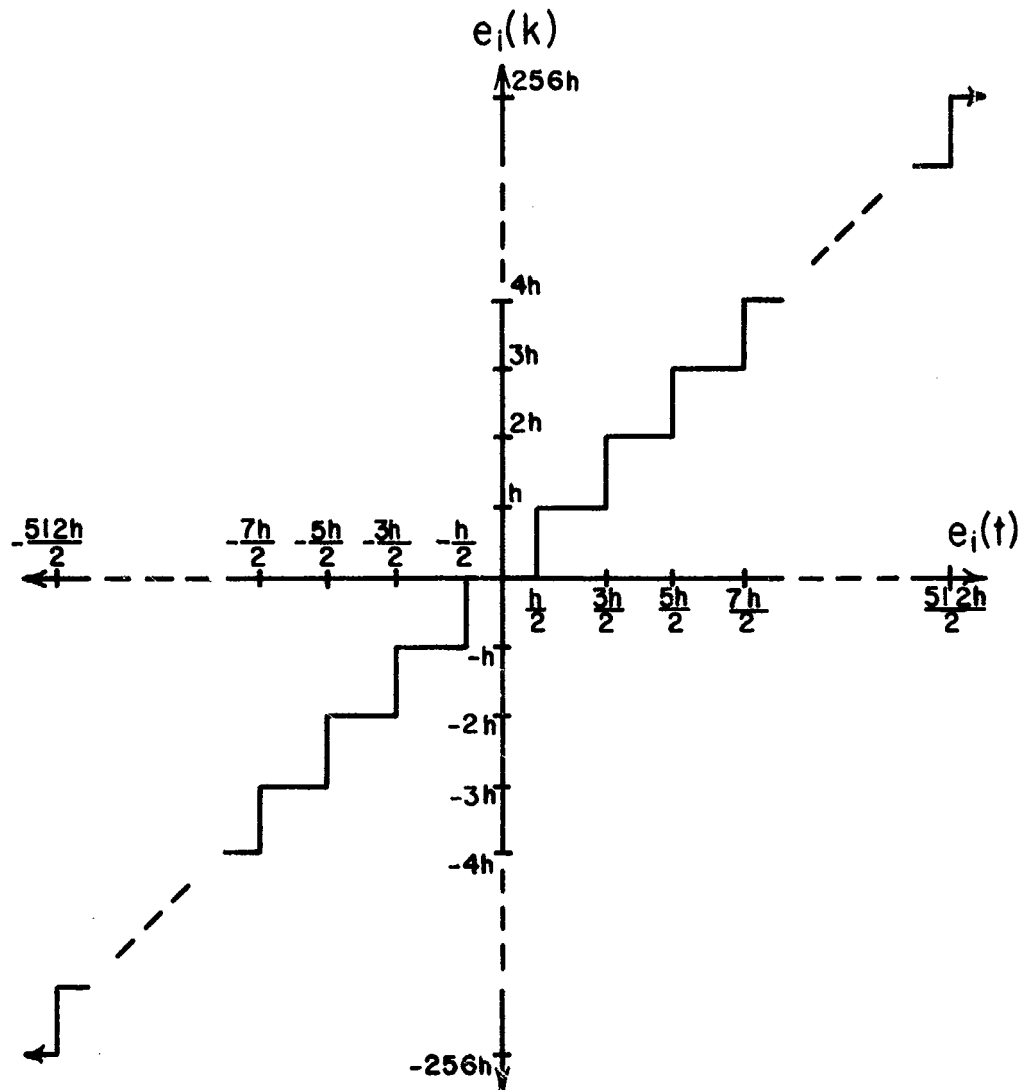


Fig. 12. Quantization characteristics of a successive approximation A/D round-off converter.

Digital-to-analog conversion.

The purpose of the digital-to-analog (D/A) converter is to convert the quantized digital output word $e_{oq}(k)$ of Fig. 6 into the analog voltage $e_{oq}(t)$. Because of the conversion of one type signal into another by the D/A, the D/A is known as the output interfacing element.

The type D/A used depends on the system in which it is being used. For both the PIGA loop and the Saturn V t.v.c. system, the D/A used will have to convert an eight-bit signed magnitude digital word into a continuous analog signal. The digital output word that is inserted into the D/A originated from the output storage register, a part of the Memory discussed earlier. The word is in signed magnitude form, since this form is required by the computers Arithmetic Unit.

Honeywell manufactures D/A logic modules which possess four bit resolution that can be interconnected to give 16-bit, bipolar operation [11]. The necessity of having a 16-bit D/A converter is explained in the next section of this chapter. A simplified schematic diagram of this form is shown in Fig. 13. As can be seen from the figure, four μ -Pac D/A modules (CE071) can be used to construct this D/A form. One must note that the input to this D/A is not in signed magnitude form, hence making the necessity of a code translator which will interface the signed-magnitude digital output of the output storage register with the D/A.

The code translator shown in Fig. 14 is a circuit which simply coincidizes each bit of the output storage register with the sign bit, resulting in a bipolar D/A code that can be used by the CE071 D/A

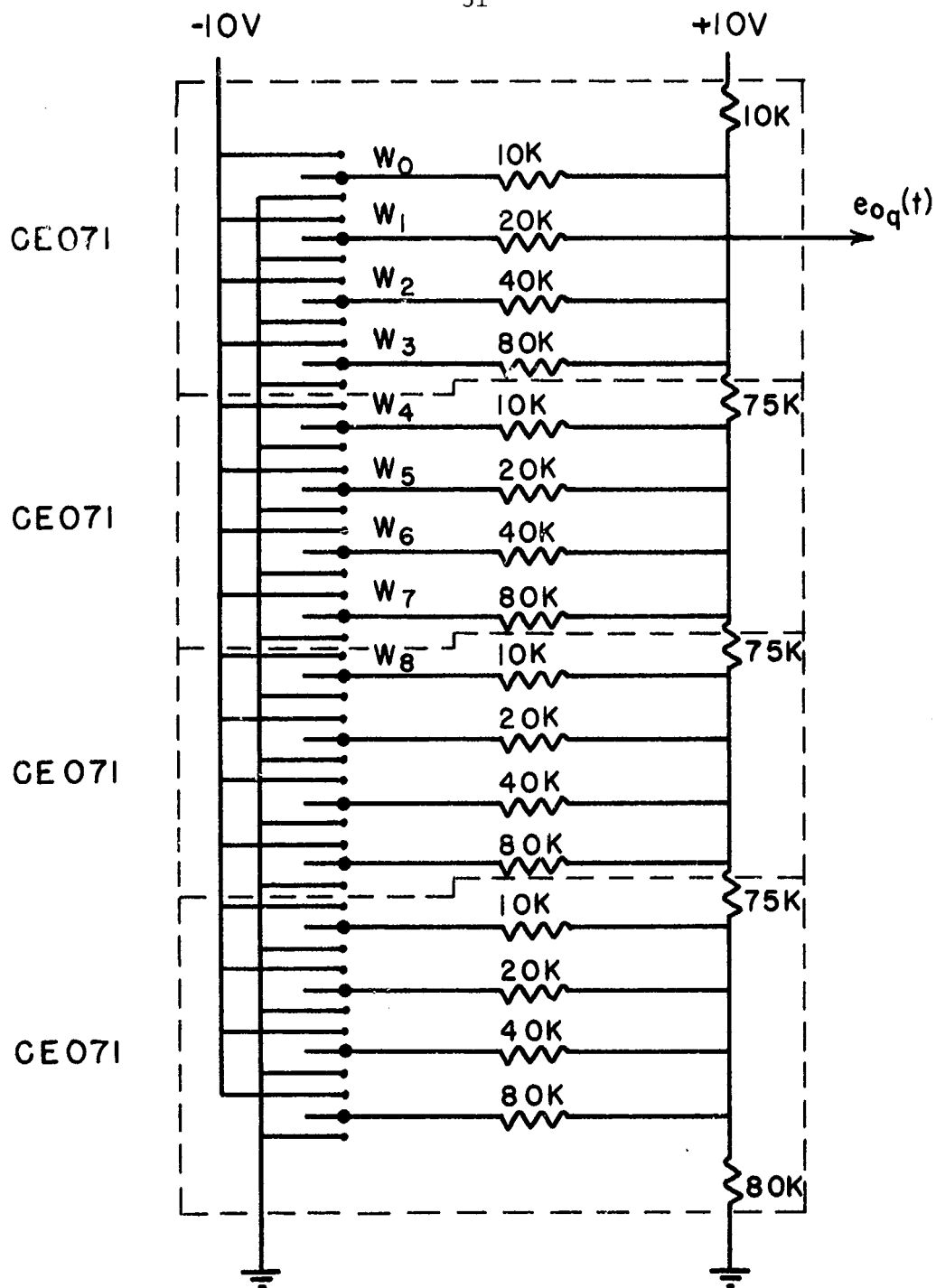


Fig. 13. A bipolar-voltage D/A constructed from Honeywell u-Pac logic modules.

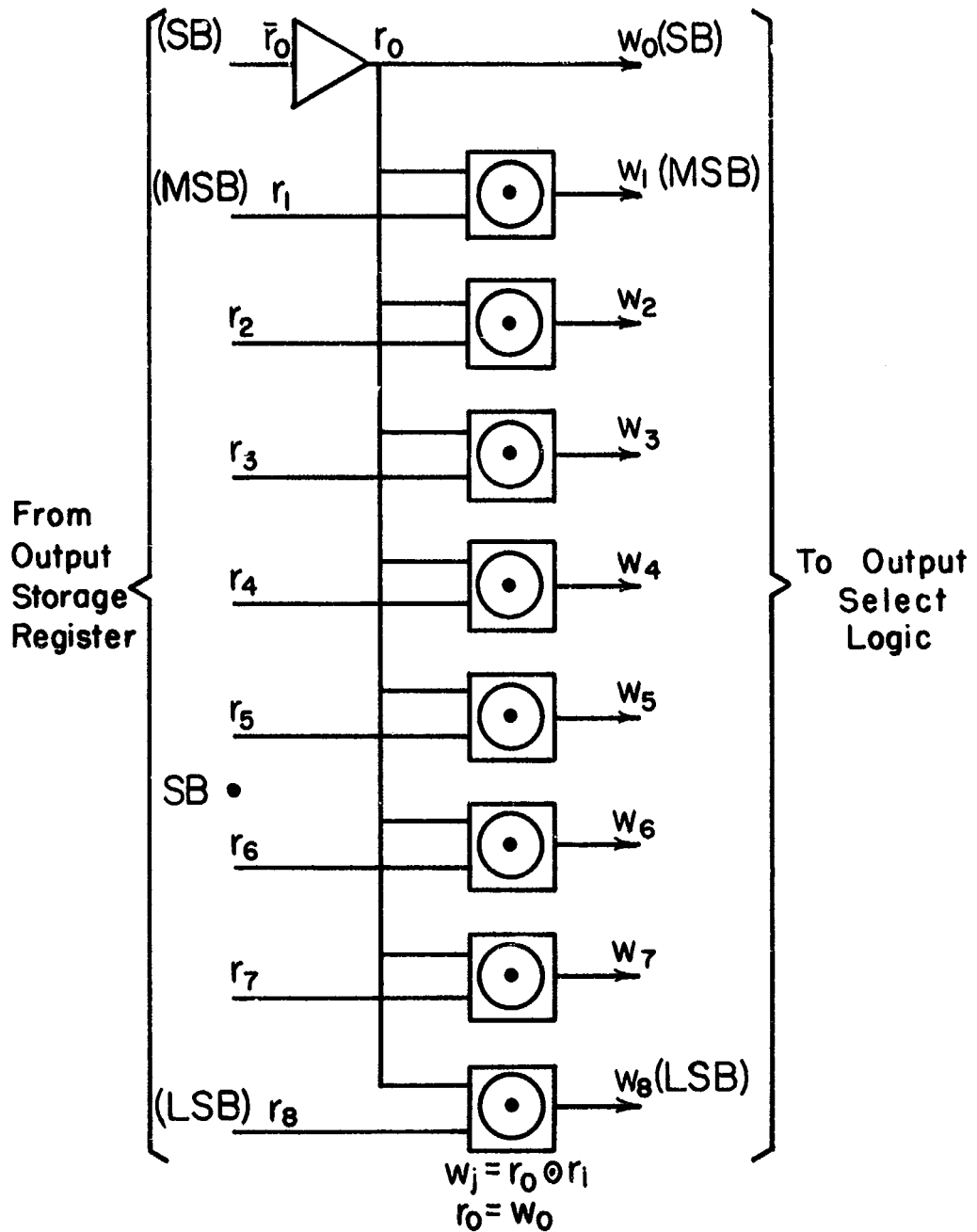


Fig. 14. Code translation logic for bipolar D/A.

modules. A coincidence gate operates in the following manner: when the sign bit and the magnitude bit are the same, the output of the gate is a logic "one" and when the sign-bit and the magnitude bit are different, the output of the gate is a logic "zero." A code truth table of the code translator is shown in Table 2.

Input and output select logic.

It is the purpose of the input and output select logic to determine the range of operation of the computer each sampling period. In doing so the input digital word to the computer is quantized to obtain $e_{iq}(k)$ of Fig. 6 and the output digital word $e_{oq}(k)$ is properly weighted before being changed to the analog signal $e_{oq}(t)$. A logic diagram of the input and output select logic is shown in Fig. 15.

The operation of the input and output select logic can be explained as follows. The purpose of the input select logic is to select from an eight bit digital word four bits with which the computer can word with each sampling period. The input select logic inserts the four most significant bits, A_{16} through A_{128} of Fig. 15, into the computer whenever anyone of these bit positions is a logic "one" and the four least significant bits, A_1 through A_8 of Fig. 15, whenever the four most significant bits just mentioned are all logic "zero." By doing this the original eight bit digital approximation of the analog input $e'_1(t)$ is quantized with the four most significant bits having a weight of sixteen times the four least significant bits.

As a numerical example of the quantization technique, assume that the digital output of the A/D of Fig. 11 is as shown below.

SIGNED-MAGNITUDE CODE	BIPOLAR D/A CODE	D/A OUTPUT VOLT.
r_0 r_1 r_2 r_3 ----- r_8	w_0 w_1 w_2 w_3 ----- w_8	$e_{oq}(t)$
1 1 1 1 1	1 1 1 1 1	$-(N-1)V/N$
1 1 1 1 0	1 1 1 1 0	$-(N-2)V/N$
1 0 0 0 1	1 0 0 0 1	$-V/N$
1 0 0 0 0	1 0 0 0 0	0
0 0 0 0 0	0 1 1 1 1	V/N
0 0 0 0 1	0 1 1 1 0	$2V/N$
0 1 1 1 0	0 0 0 0 1	$(N-1)V/N$
0 1 1 1 1	0 0 0 0 0	$V=10/3$ Volts
		$N=512$

Table 2. Code truth table.

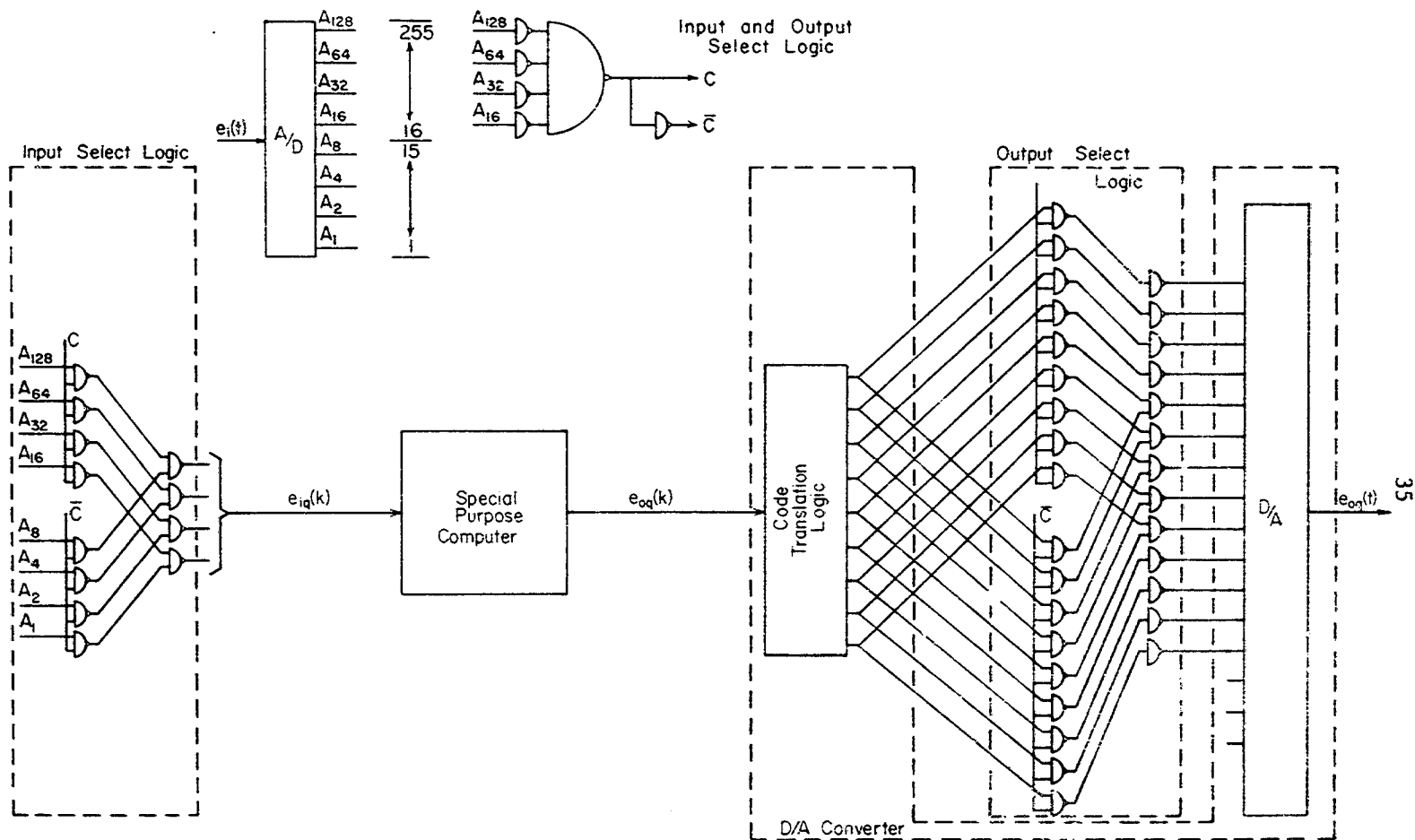


Fig. 15. Input and output select logic.

A_{128}	0	(MSB)
A_{64}	1	
A_{32}	1	
A_{16}	1	
A_8	0	
A_4	1	
A_2	0	
A_1	1	(LSB)

This digital approximation has a numerical value of $(117)_{10}$, but because of the conditions described earlier, only the four most significant bits, $(0111)_2$, with a numerical value of $(112)_{10}$ are inserted into the computer. It can be seen that because of the quantization technique, the original digital approximation of $e_1(t)$ has been reduced by five. Now suppose that for another sampling period, the digital output of the A/D is as shown below.

A_{128}	0	(MSB)
A_{64}	0	
A_{32}	0	
A_{16}	0	
A_8	0	
A_4	1	
A_2	0	
A_1	1	(LSB)

In this case the four most significant bits are disregarded and the

digital word $(0101)_2$, which has a numerical value of $(5)_{10}$, is inserted into the computer. The digital word $(0101)_2$ has a weight of $1/16$ of what it would have had if it had been in the four most significant bit positions at the output of the A/D.

It is because of the input quantization technique that the internal variables $m_q(k-1)$ and $m_q(k-2)$ of Fig. 6 have to be shifted up and down (properly weighted) each sampling period as described earlier. This is done so that each sampling period the internal variable $m_q(k-1)$ has the same weight as $m_q(k)$ of the previous period and $m_q(k-2)$ has the same weight as $m_q(k-1)$ of the previous sampling period.

It is because of the input quantization that the output select logic is required. The purpose of the output select logic is to properly weight the bipolar coded D/A output of the output storage registers; i.e., the output select logic provides a means by which the digital output of the S-P computer has a magnitude 16 times as great when working in the "higher range" as it does when working in the "lower range." The operation of the output and input select logic coincides; that is, during a sampling period the input and output select logic both operate in the same "range." The weighting process is performed by inserting the bipolar coded D/A output of the storage register into the nine most significant bit positions of the D/A when the computer is working in the "higher range" and by inserting the output into bit positions four bits lower when operating in the "lower range." From Fig. 13, the sixteen bit D/A is used because each μ -Pac digital module has four bit positions in it, therefore making it impossible to construct

only a 13-bit D/A.

The following explanation along with Fig. 15 will help to make it clearer how the input and output select logic operates during a sampling period. First, the digital approximation of $e_1^*(t)$ appears at the output of the A/D. If any of the four most significant bits are at logic "one," C will also be a logic "one," thereby enabling the four most significant nand gates and disabling the four least significant of the input select logic, thus inserting the four most significant bits into the S-P computer. Later on during this same sampling period, a digital output appears at the output storage register. As the computer is working in the "upper range" and C is a logic "one," the bipolar D/A coded output of the output storage register is inserted by the output select logic circuitry into the most significant bit positions of the D/A. For the other case when during a sampling period all the four most significant bits are at logic "zero," C is at logic "one" thereby enabling the four least significant nand-gates of the input select logic which inserts these bits into the computer. As before, later on during the sampling period a digital word appears at the output of the output storage register and since C of the output select logic is at logic "one," the bipolar coded output of the output storage register is inserted into the less significant bit positions of the D/A.

Arithmetic Unit

As stated earlier, the digital filter being designed is designed such that two discrete mathematical equations may be physically realized. The two equations that are physically realized by the S-P

computer of the filter are repeated below.

$$e_o'(k) = a_o e_{iq}(k) + c_1 m_q(k-1) + c_2 m_q(k-2) \quad (3-3)$$

$$m'(k) = e_{iq}(k) - b_1 m_q(k-1) - b_2 m_q(k-2) \quad (3-4)$$

By observing equations (3-3) and (3-4), it can be seen that an Arithmetic Unit must be designed with the capabilities of performing six multiplications and four additions each sampling period. It is the purpose of this section of the chapter to first explain how a digital adder (parallel binary accumulator) is implemented, how the multiplication process is accomplished using the digital adder and shift registers and finally the digital logic required to quantize $e_o'(k)$ and $m'(k)$ of equations (3-3) and (3-4).

Parallel binary accumulator.

It is the purpose of the parallel binary accumulator to perform the digital additions required to physically realize equations (3-3) and (3-4) above. A binary accumulator contains a register which stores a binary number called the augend in signed magnitude form and after receiving another binary number, called the addend, in the same form adds the second number to the first and then stores the sum in the register. The complete operation of the parallel binary accumulator is explained in full by [20]. A digital logic diagram of the parallel binary accumulator being discussed is shown in Fig. 16. Each of the flip-flops shown in the accumulator serves as a modulo-2 counter. The

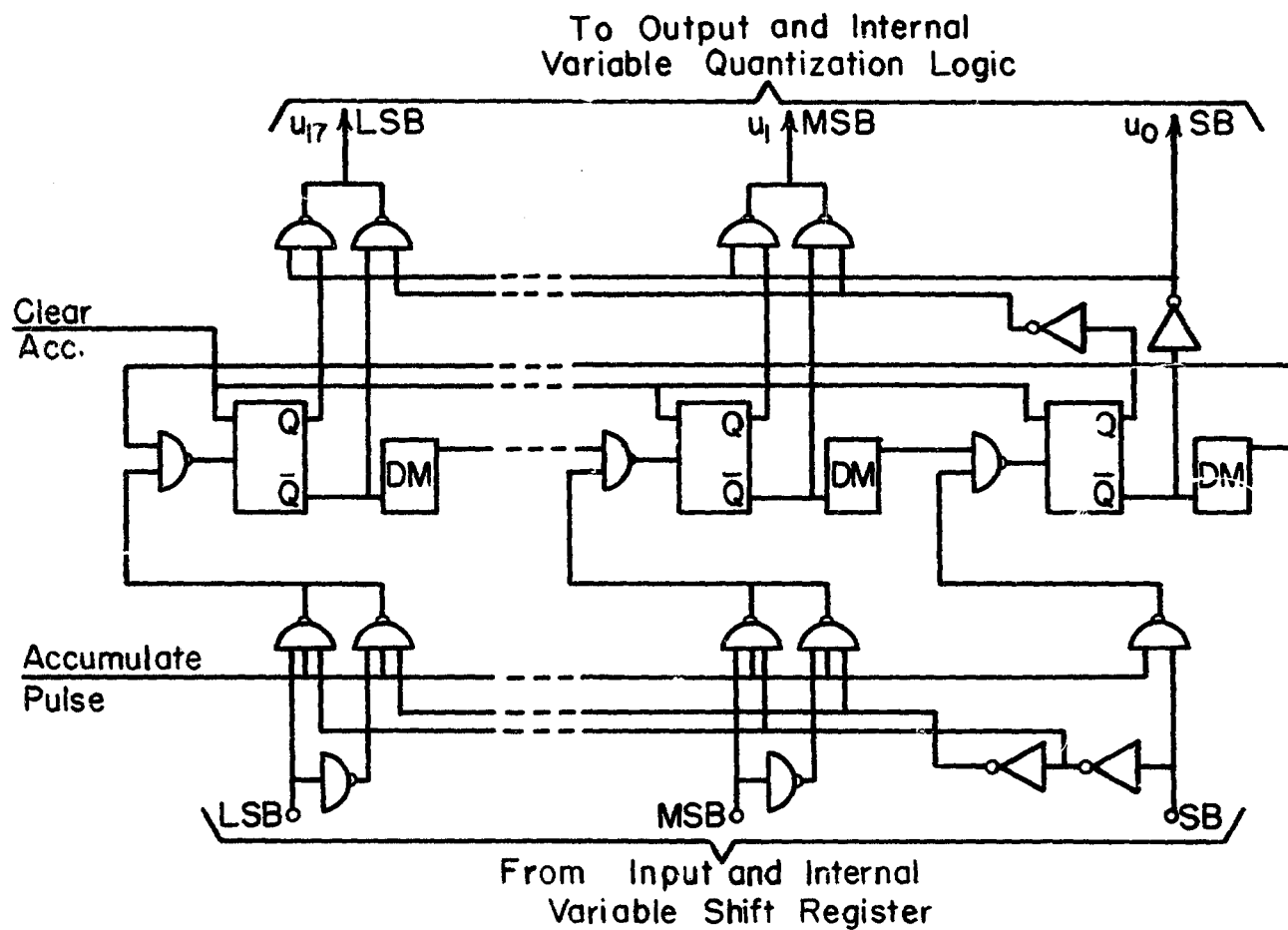


Fig. 16. Logic diagram of parallel binary accumulator.

augend is initially stored in the accumulator , and during the addition process, each flip-flop counts parallel incoming pulses representing the addend bits and generates a carry pulse to the next significant bit when the flip-flop changes state from logic "one" to "zero."

In constructing the parallel binary accumulator it is important to note that a carry-pulse propagation time-delay interval will result. It is important that this time-delay interval is known since it regulates how fast inputs into the accumulator are entered. As an example of how a carry pulse propagation time delay interval will result, assume that a carry pulse is generated in the least significant bit of the accumulator. If this carry pulse generates a second carry pulse at the next-higher significant bit, and if this chain reaction continues through the whole accumulator, the time-delay interval mentioned above results.

To determine the total carry propagation time of the accumulator, the carry propagation time will be calculated for each stage and this number multiplied by n , the number of stages contained in the accumulator. The propagation time for one stage of Fig. 16 is the time-delay through one nand gate (20ns), one flip-flop (60 ns), and one delay multivibrator (60 ns). When these time-delays are summed, the delay time for one stage is found to be 240 ns. The accumulator in Fig. 16 is designed as an 18 bit accumulator, so the total carry propagation time is $4.32 \mu \text{ sec}$. This now being known, the maximum rates at which inputs can be entered into the accumulator (230 KHz) is also known.

To determine the size of the accumulator, the maximum possible

values of equations (3-3) and (3-4) were calculated using the coefficients necessary for satisfactory operation of the PIGA loop and the Saturn V t.v.c. system. It was found that the maximum magnitude of a number the accumulator would contain was $(336)_{10}$, thus making it necessary for nine bits to the left of the binary point. The requirement of 8-bits to the right of the binary point will be explained in the next section of this chapter. The nine bits to the left of the binary point, plus eight bits to the right of the binary point and a sign bit constitute the eighteen bit accumulator.

Shift registers.

Now that a method has been devised to add signed magnitude binary numbers, a multiplication process must be devised for these numbers. By examining any real number which is not an integer, one can see that it can be written as an integer plus a sum of fractions. As an example, take the number 1.875. This number can be represented as an integer plus a summation of fractions as $1 + 1/2 + 1/4 + 1/8$ and in the binary number system as $(1.111)_2$. In the binary number system, multiplication of a binary number by real number less than one is done by moving the binary point to the left. Moving the binary point one place to the left multiplies the number by $(1/2)_{10}$, moving two places to the left multiplied the number by $(1/4)_{10}$ and in general, moving the binary point n places to the left multiplies the number by $1/2^n$. As an example, take the number $(16)_{10}$ represented in binary form as $(10000)_2$. This number can be multiplied by $(1/2)_{10}$ by moving the binary point one place to the left to get $(8)_{10}$ which is represented in binary as

$(1000.0)_2$ or it can be multiplied by $(1/4)_{10}$ by moving the decimal point two places to the left as shown $(100.00)_2$ to obtain $(4)_{10}$.

Conversely, if the decimal point is moved to the right n places, the binary number is multiplied by $(2^n)_{10}$. The above information is useful when devising a multiplication scheme for binary numbers. Looking at the above information, one might see that multiplication in binary can be accomplished by a shifting of the binary point and summation process. As an example suppose that the multiplication of $(16)_{10}$ by $(1.75)_{10}$ is desired. This can be represented as

$$(1 + 1/2 + 1/4)_{10} (10000.)_2$$

which is the same as

$$(1)_{10} (10000.)_2 + (1/2)_{10} (10000.)_2 + (1/4)_{10} (1.0000.)_2,$$

or after the multiplications have been carried out by shifting the binary point, the following results

$$(10000.)_2 + (1000.)_2 + (100.)_2 = (11100.)_2 = (28)_{10}.$$

From this it is seen that multiplication can be done by a shifting of the binary point and adding process. In the S-P computer the shifting process will be done by the shift registers and the adding is done by the accumulator. As an example of how the multiplication process is performed in the S-P computer, the decimal number eight will be multiplied by $(1.25)_{10}$. This is represented in binary form as $(1000.)_2 (1.01)_2$.

Since the 1 bit position of $(1.01)_2$ has a "one" in it $(1000.)$ will be inserted into the accumulator. Next the binary point is shifted one place to the left to multiply by $(1/2)_{10}$ but since this bit position is "zero" in $(1.01)_2$, $(100.)_2$ will not be inserted into the accumulator. Now the binary point is shifted one more place to the left to multiply the number by $(1/4)_{10}$ and $(10.)_2$ results. Since the $(1/4)_{10}$ bit position is at logic "one," $(10.)_2$ will be inserted into the accumulator. Now the numbers inserted into the accumulator have been added since the accumulator keeps a cumulative total of all numbers inserted into it and as a result the accumulator contains the result of the multiplication which is represented at the output of the accumulator in binary as $(1010.)_2$. This process might be understood more clearly if it is shown as below.

+	1000.	
X	<u>1.01</u>	
+	1000.	multiply by 1
+	000.0	multiply by 0
+	<u>10.00</u>	multiply by 1/4
+	1010.00	

As earlier stated the shifting of the decimal point will be accomplished by the shift registers. Actually the decimal point remains in the same position and the bits are shifted. A logic diagram of the shift registers is shown in Fig. 17. There are actually two sets of shift registers required, one the coefficient register which is eight stages long including the sign bit and the other the internal variable register which including the sign bit, is fifteen stages long. The shift register shown in Fig. 17 has information entered into it in parallel via the DC set terminals. Each individual

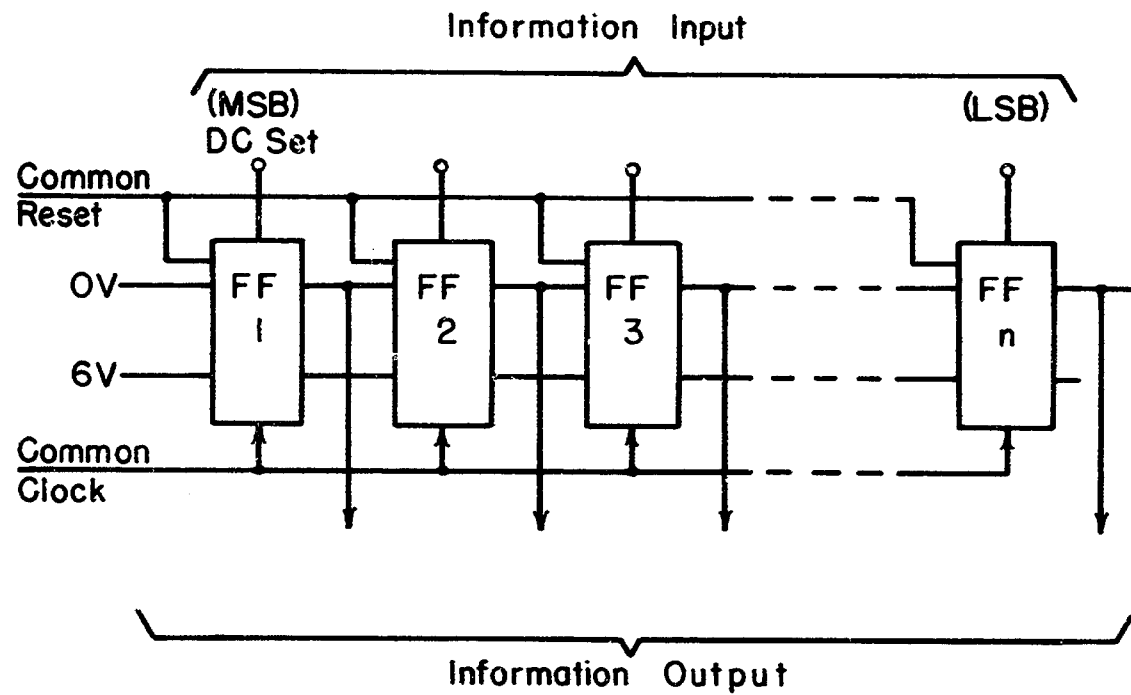


Fig. 17. Shift registers of the Arithmetic Unit.

f-f of the register can be set to a logic "one" if a 1-0-1 signal is applied to its DC set terminal. Information is shifted to the right by applying 0-1-0 pulses to the common clock terminal. Each 0-1-0 pulse will shift the number in the register one place to the right. Last, the shift register can be cleared by the application of a 1-0-1 pulse to the common reset terminal.

In explanation of how the two shift registers and the parallel binary accumulator are combined to perform the multiplication process, Fig. 18 might be helpful. The output of an internal variable storage register or the input, and a coefficient are all entered into their respective shift registers first by a command generated by the control circuitry. Next alternating accumulate and shift pulses are applied to the accumulator and shift registers respectively by the control circuitry to perform the multiplication process. After this process is completed, the result of the multiplication appears at the output of the accumulator where it is then inserted into the output storage register or the $m_q(k-1)$ internal variable storage register. The proper sign that accompanies the multiplication process is achieved by the use of the "exclusive-or" gate shown in Fig. 13. If the two sign inputs are the same (using "0" for positive and "1" for negative) the output is logic "0" or positive and if the sign inputs are different, the output of the exclusive-or gate is a logic "1" which is negative.

Reduction logic.

As earlier stated, the compensator possesses the feature of having quantization at three points, the first at the output of the

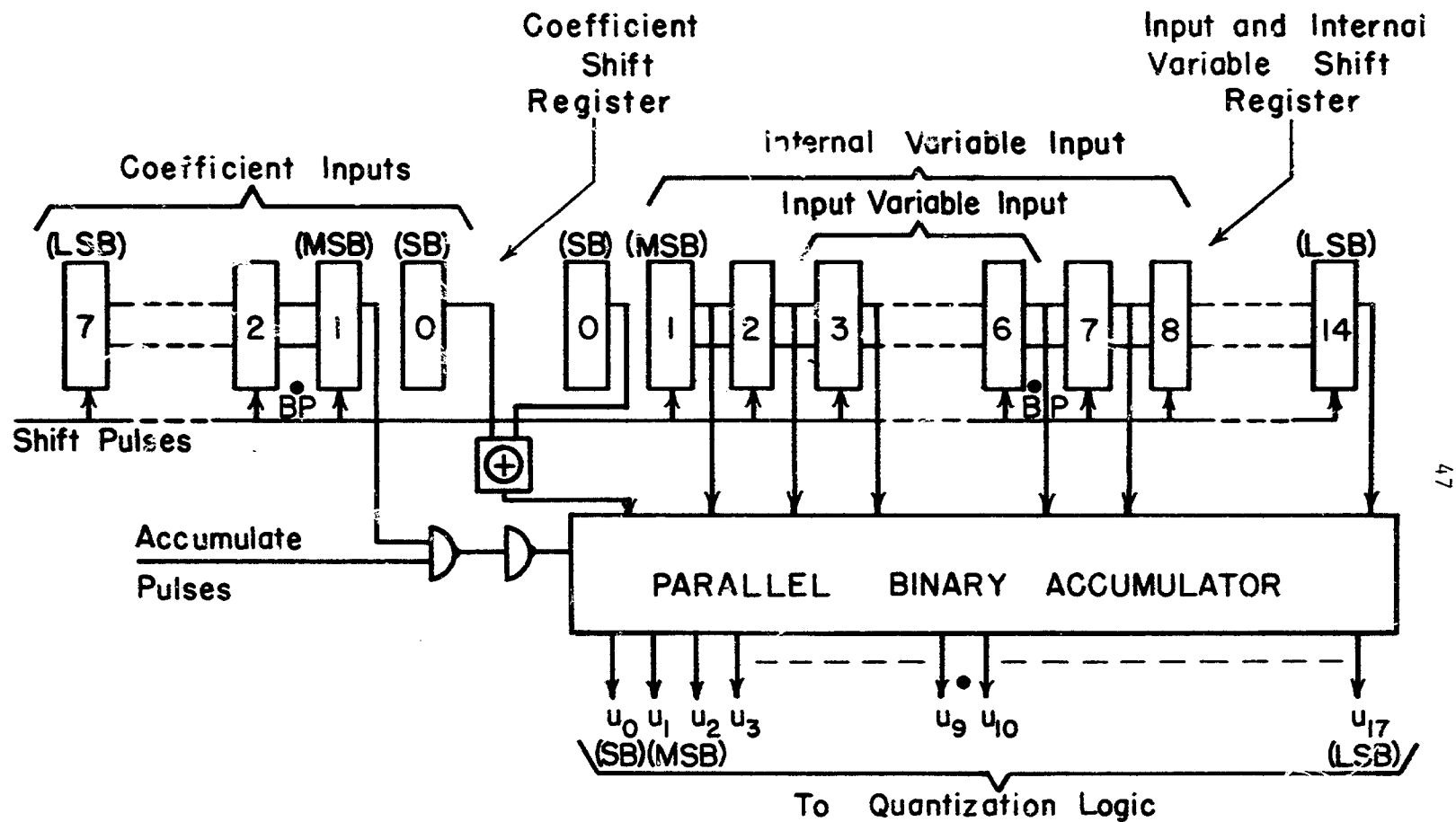


Fig. 18. Logic diagram of shift registers and accumulator combined to perform the multiplication process.

A/D converter and the other two at the filter output and internal to the filter. The quantization at the output of the filter is achieved by quantizing the digital output of the S-P computer before it goes to the D/A converter and the internal quantization is performed by quantizing $m'(k)$ before it is entered into the internal variable storage register $[m_q(k-1)]$. The quantization at these two points is simply a round-off process. For the output quantization, $e_o'(k)$ of Fig. 6 as it emerges from the accumulator can contain a maximum magnitude of eight bits to the right of the binary point and eight magnitude bits to the left of the binary point. It is the purpose of the output quantization logic to round-off this digital number such that it can have a maximum of five bits to the left of the binary point and three bits to the right of the binary point, making the magnitude have a maximum value of $(31.875)_{10}$. Likewise, the internal variable $m'(k)$ can have a maximum of nine magnitude bits to the left of the binary point and eight to the right of the binary point as it emerges from the accumulator. This digital word is quantized such that it has a maximum of six bits to the left of the binary point and two bits to the right of the binary point, making it possible for it to have a maximum magnitude of $(63.75)_{10}$.

The best quantization levels at these two points was determined by running the Fortran source program shown in the appendices which simulated the PIGA loop and the Saturn V t.v.c. system. Several different quantization levels were tried in the programs and it was found that the levels used achieved the best desired responses from both

systems.

A digital logic diagram of the two quantizers discussed above is shown in Fig. 19. Here the output of the accumulator goes directly to both quantizers. The output of the quantizer which reduces the output of the accumulator to five bits to the left of the binary point and two to the right is first clocked by the controller into the output storage register and then later in time during the same sampling period the output of the other quantizer is clocked into the $m_q(k-1)$ storage register, thus making it possible for the output of the accumulator to go directly to both quantizers. Observing the digital logic of the quantizers, it is seen that if any one of the most significant bits being quantized (the four most significant bits for the output quantizer and the second, third, and fourth most significant for the internal variable quantizer) is a logic "one," all the output bits of the quantizers are at logic one. This is done so that the output of the quantizers will have a magnitude which is closer to the actual magnitude of the output of the accumulator. As an example, suppose that the output of the accumulator which is to be quantized by the output quantizer is $(00100000.000)_2$ which is $(32)_{10}$. If a quantization scheme were used where the three most significant bits were eliminated and the remaining bits inserted into the output storage register, the input to the storage register would be $(00000.000)_2$ which is $(0)_{10}$. As can be seen there is a large difference in the magnitude of the output of the accumulator and the magnitude of the number being inserted into the output storage register. Because of the digital logic

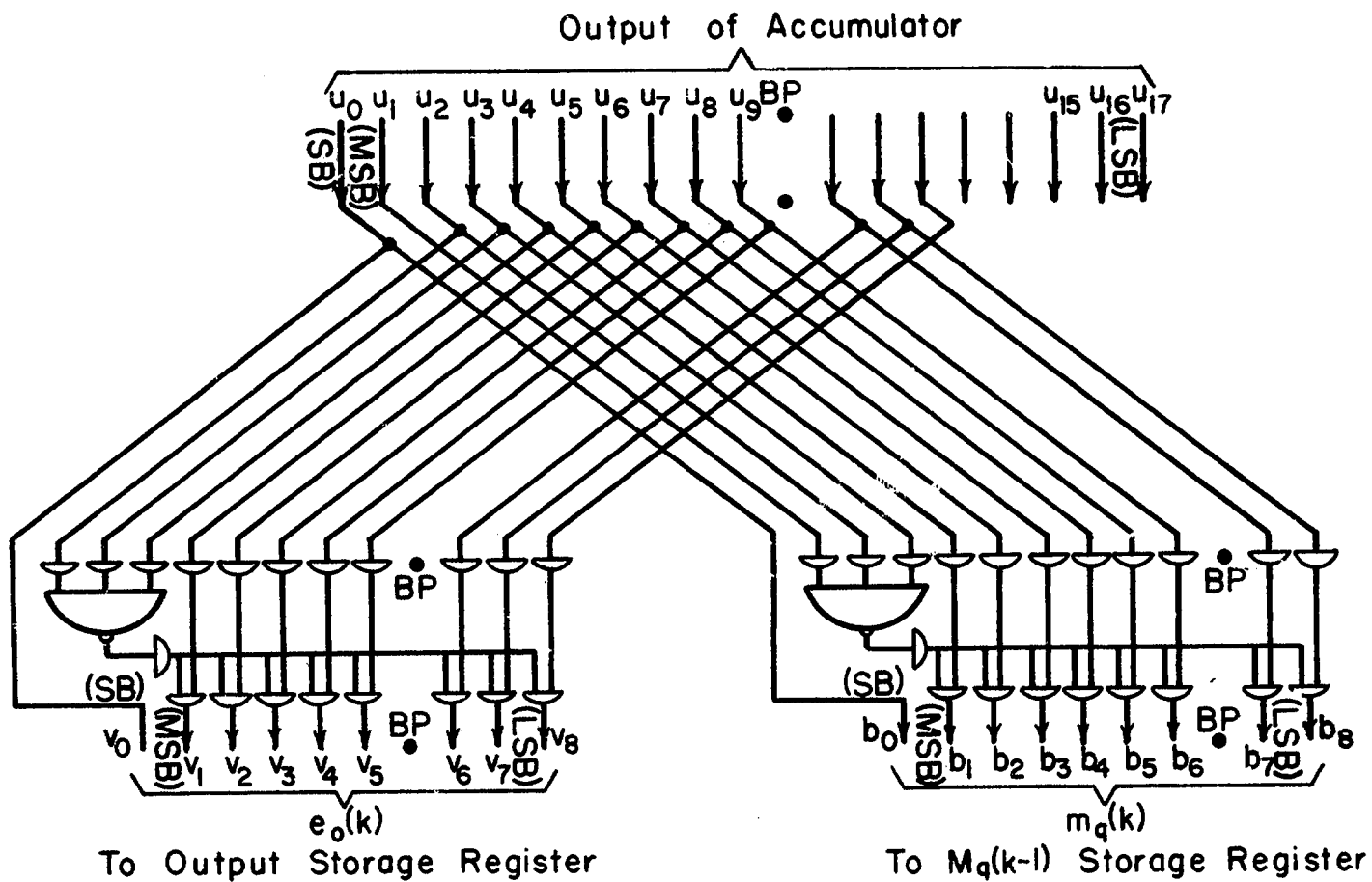


Fig. 19. Logic diagram of quantizers.

employed in Fig. 19, for the same output of the accumulator as before, the input to the output storage register is now $(11111.111)_2$ which is $(31.875)_{10}$; this number being much closer to the actual magnitude of the output of the accumulator than before.

Memory

Each of the blocks in Fig. 9 performs a certain function in relation to the over-all operation of the digital compensator. The purpose of the Memory is to perform the time delay function of the difference equations (3-3) and (3-4), to provide a storage location for the variables $m_q(k-1)$, $m_q(k-2)$, $e_{oq}(k)$ and to provide a storage location for the a_0 , c_1 , c_2 , b_1 , and b_2 coefficients. The storage elements can be divided into three groups; first, the output storage; second, the internal variable storage and last, the coefficient storage. The above three groups of storage elements will be discussed fully in the order as they appear above.

Output storage.

It is the purpose of the output storage element to provide a location at which the quantized digital output, $e_{oq}(k)$, of the S-P computer can be held while being changed to an analog voltage level by the code translator and the D/A converter. The logic diagram of the output storage element is shown in Fig. 20. The storage register is constructed from buffer register flip-flops contained on Honeywell μ -Pac BR335 digital modules. Information from the output quantization logic is entered into the storage register in parallel through the set control terminals. A 0-1-0 pulse applied to the common clock

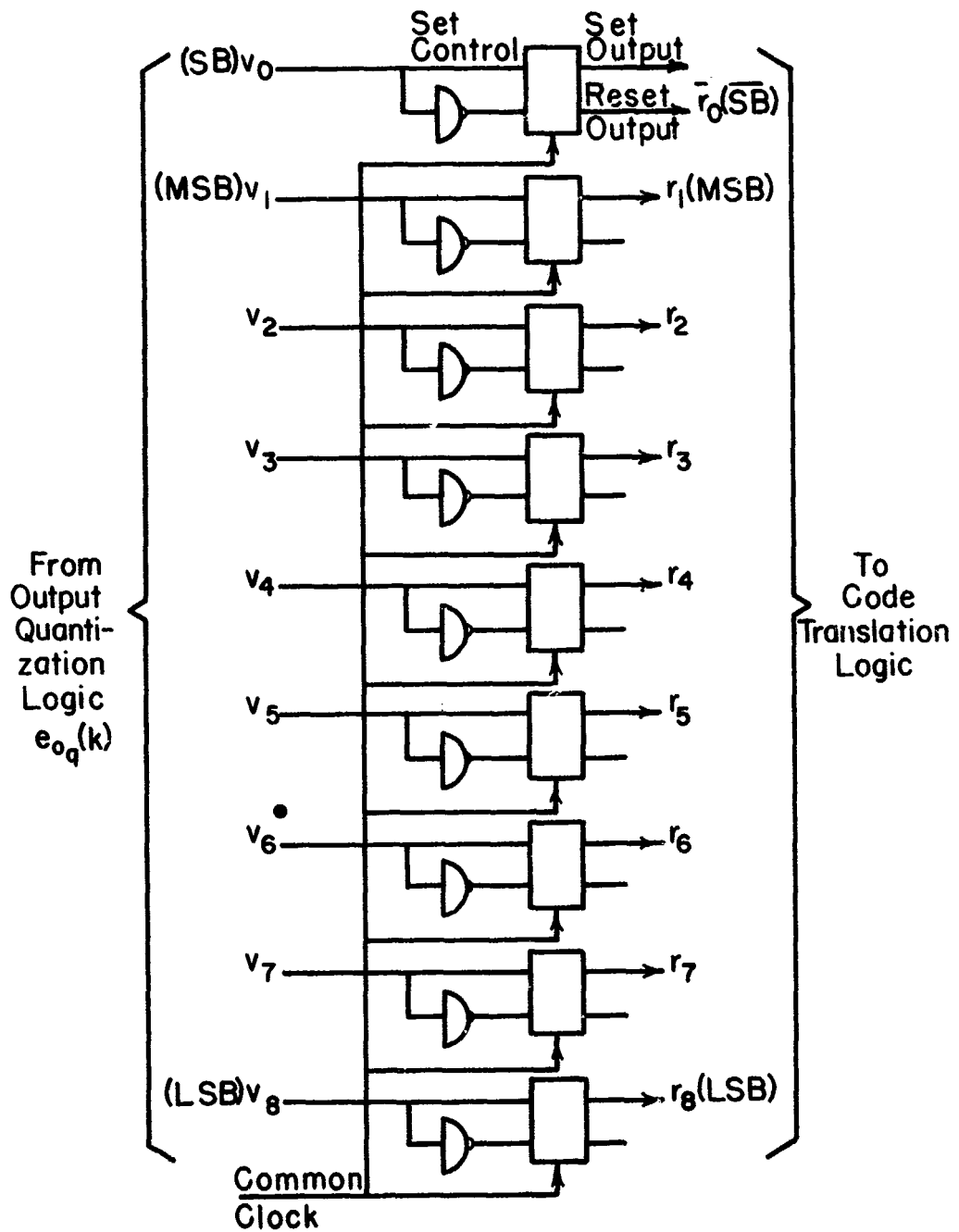


Fig. 20. Output storage register.

terminal at the time during the sampling period just after $e_{oq}(k)$ is calculated transfers the information contained at the set control terminals to the set output terminals. Here, $e_{oq}(k)$ is held so that the code translation logic and the D/A converter can convert $e_{oq}(k)$ into the quantized analog output $e_{oq}(t)$. $e_{oq}(k)$ is held at the set output terminals until the next sampling instant. Then a new $e_{oq}(k)$ is calculated and clocked to the set output terminals. From Fig. 20, it is seen that the complement of the sign bit is desired as the input of the code translation logic. This is necessary for proper operation of the code translation logic as can be seen from Fig. 14.

Internal variable storage.

To physically realize equations (3-3) and (3-4), it is seen that the S-P computer must have a way in which the internal variables can be delayed in time and at the same time stored. This is accomplished by the use of the internal variable storage registers, $m_q(k-1)$ and $m_q(k-2)$. From earlier discussion of the "range switching" process, these two storage registers must be able to shift their contents on command down four positions, up four positions, or in parallel from one storage register to the other. How this is done will be explained with the aid of Fig. 21.

If the S-P computer has been operating in the "lower range" and then during a sampling period it changes to the "higher range" of operation, it will be necessary, as explained earlier, for the information in the storage registers to be shifted down four bit positions. This is done by letting S of Fig. 21 be logic "zero," D logic "one,"

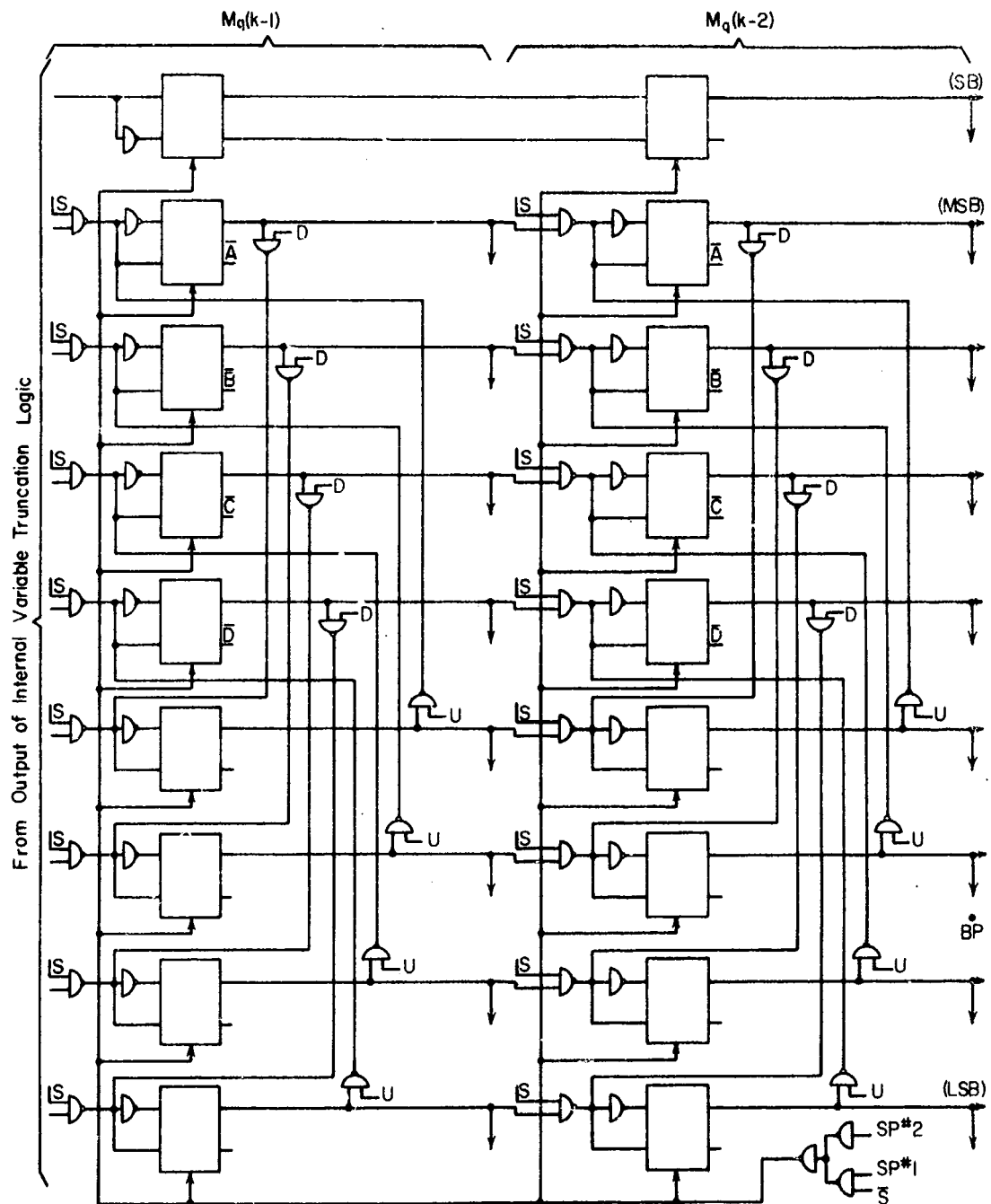
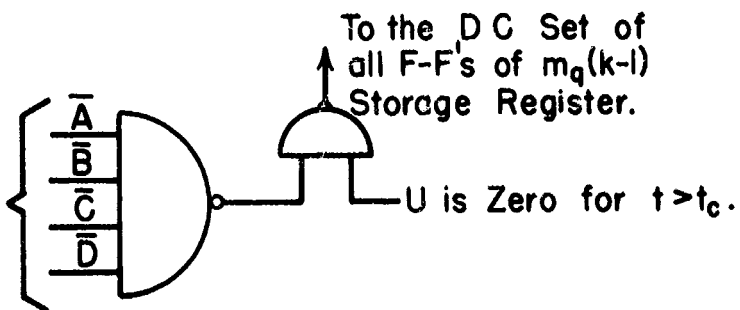


Fig. 21. Shift "up" or "down" logic of internal variable storage registers.

and U logic "zero" and then the application of shift pulse #1 shifts the information down. Now taking the case when the computer has been operating in the "higher range" and changes to the "lower range" of operation, the information contained in the registers must be shifted up four positions. If a logic "one" is in any one of the four most significant bit positions before shifting, a logic "one" is inserted into each f-f of the storage register by the saturation logic shown in Fig. 22. From Fig. 21, the shifting up operation is done by letting S be at logic "zero," D at logic "zero," and U at logic "one." While the just mentioned terminals are at their respective logic, shift pulse # 1 is applied to shift the information. If the S-P computer is operating in the same range each sampling period, only parallel shifting of the information is required. This is done by letting terminals D and U remain at logic "zero," S at logic "one" while shift pulse # 2 is applied to the common clock terminals of the storage registers.

The S, D, and U terminals of Fig. 21 all have the proper logic when shift pulse # 1 or 2 is applied because of the "range" memory logic shown in Fig. 23. The "range" memory logic is a single f-f. The "range" the computer is working in during the present sampling period is contained at the set control terminal of the f-f and the "range" of operation of the previous sampling period is contained at the set output terminal of the f-f. A storage element of the present and the previous "range" of operation is required so that the computer will know when to shift the information contained in the internal variable

From Reset
Outputs of Four
Most Significant
Bit Positions of
 $m_q(k-1)$ Storage
Register.



From Reset
Outputs of Four
Most Significant
Bit Positions of
 $m_q(k-2)$ Storage
Register.

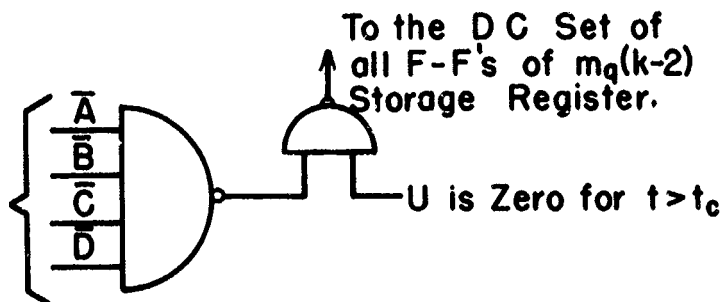


Fig. 22. Saturation logic.

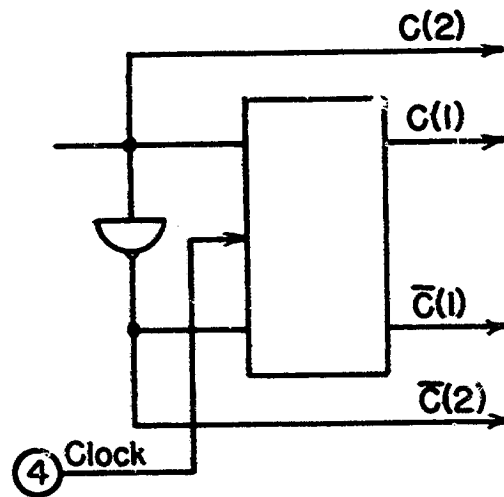
C(2)	C(1)	S	D	U
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

$$U = \bar{C}(2) \cdot C(1)$$

$$D = C(2) \cdot \bar{C}(1)$$

$$S = C(2) \odot C(1)$$

Shift Logic Truth Table



Range Memory

Shift Logic Circuitry

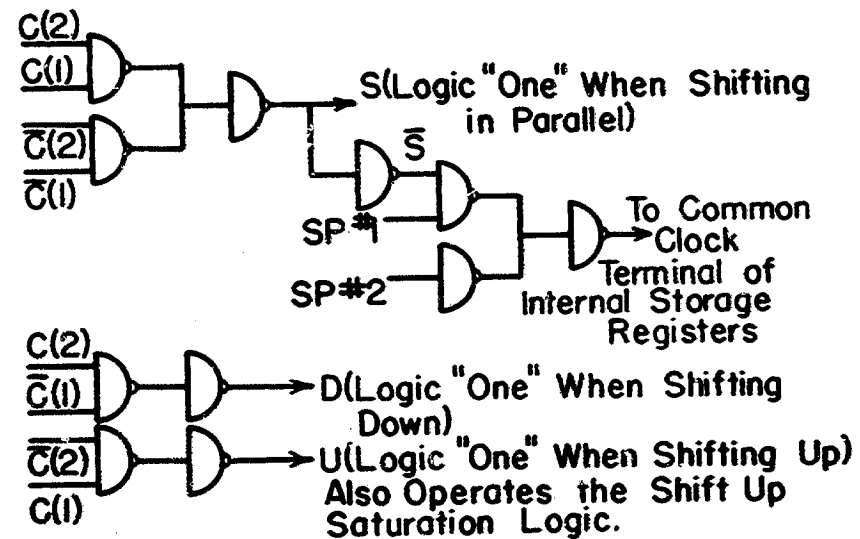


Fig. 23. Logic diagram of "range" memory and associated circuitry.

storage registers up or down. Along with the "range" storage element and the logic circuitry implemented from the truth table of Fig. 23, the proper logic at the S, D, and U terminals of Fig. 21 is always assured.

It is seen from the above discussion that two common clock pulses are applied to the storage registers during one sampling period of operation. The first shift pulse (SP. # 1) does the up or down shifting of the storage elements and the second shift pulse (SP. # 2) does the parallel shifting of the information contained in the storage elements. The sequence of application of these two pulses will be explained in a following section of this chapter.

Coefficient storage.

From equations (3-3) and (3-4) it is seen that there must be a method designed by which the a_0 , c_1 , c_2 , b_1 , and b_2 coefficients can be stored since they are used repeatedly by the S-P computer in calculating the above mentioned equations. The coefficients are stored by the use of single-pole double throw switches as shown by Fig. 24. By simply toggling the switch back and forth, a logic "one" (six volts) or a logic "zero" (ground) may be stored for an infinite length of time. One switch is used for each bit of each coefficient, i.e., if a coefficient contains seven binary bits plus a sign bit, eight switches will be required to store this coefficient. How the coefficients are transferred from the Memory to the Arithmetic Unit when needed will be discussed in the next section of this chapter.

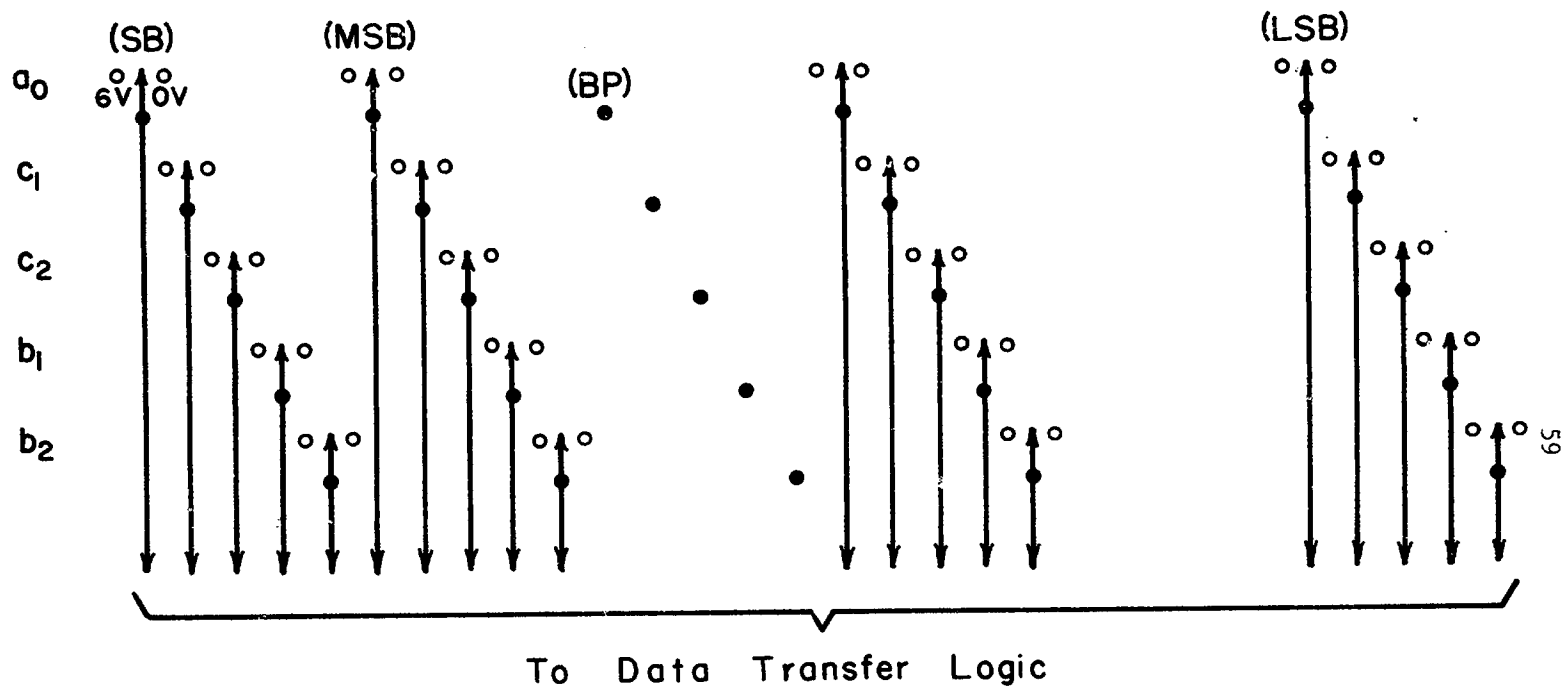


Fig. 24. Coefficient storage.

Control Unit

Now that all the components of the digital compensator have been designed and digitally implemented, a Control Unit must be designed which co-ordinates the operation of the components when working together to physically realize equations (3-3) and (3-4). The Control Unit will be divided into two units, the control function generator and the data transfer logic. Each of the two units is discussed in full below.

Control function generator.

In designing the control function generator, an operation sequence for the compensator must be decided upon. The operation sequence is mainly determined by the operations required to realize equations (3-3) and (3-4). Either one of the equations may be realized first, but since the output of the compensator is desired as soon as possible after the input signal has been sampled by the A/D, equation (3-3) will be realized first. Once the operation sequence of the compensator has been decided upon, a timing pulse diagram can be drawn of the control pulses needed to ensure the particular sequence of operation chosen.

The operation sequence chosen for the compensator is shown below:

- (1) Activate the A/D so that $e_1(k)$ can be obtained.
- (2) Shift the $m_q(k-1)$ and $m_q(k-2)$ storage registers.
- (3) Change the "range" of the internal variable storage registers if needed.
- (4) Shift the "range" memory flip-flop.
- (5) Clear the shift registers and the accumulator.
- (6) Load $m_q(k-1)$ and c_1 into the shift registers.
- (7) Accumulate, shift, accumulate, shift, etc. until the accumulator output is $c_1 m_q(k-1)$
- (8) Clear the shift registers.
- (9) Load C_2 and $m_q(k-2)$ into the shift registers.
- (10) Accumulate, shift, accumulate, shift, etc. until the accumulator output is $c_1 m_q(k-1) + c_2 m_q(k-2)$.
- (11) Clear shift registers.

- (12) Load a_0 and $e_{iq}(k)$ into the shift registers.
- (13) Accumulate, shift, accumulate, shift, etc., until the output of the accumulator contains $a_0 e_{iq}(k) + c_1 m_q(k-1) + c_2 m_q(k-2) = e_o'(k)$.
- (14) Load $e_o'(k)$ into output storage register so that it can be changed to an analog voltage level.
- (15) Clear the accumulator and clear the shift registers.
- (16) Load $m_q(k-1)$ and $-b_1$ into the shift registers.
- (17) Accumulate, shift, accumulate, shift, etc., until the output of the accumulator is $-b_1 m_q(k-1)$.
- (18) Clear the shift registers.
- (19) Load $m_q(k-2)$ and $-b_2$ into the shift registers.
- (20) Accumulate, shift, accumulate, shift, etc., until the output of the accumulator is $-b_1 m_q(k-1) - b_2 m_q(k-2)$.
- (21) Clear the shift registers.
- (22) Load $e_{iq}(k)$ and 1.0 into the shift registers.
- (23) Provide one accumulate pulse. Now the output of the accumulator is $e_{iq}(k) - b_1 m_q(k-1) - b_2 m_q(k-2) = m'(k)$.

From the above operation sequence, a timing diagram can be drawn which shows the type pulses that will have to be generated by the control function generator and when in time they will have to be generated in relation to one sampling period. It is from this timing diagram that the control function generator is designed. Fig. 25 shows the timing and pulse diagram for the second order compensator being designed.

From Fig. 25, it is seen that the control pulses from the control function generator must be somewhat cyclic, each cycle being lettered A through G in the figure. This is an aid in designing the control function generator in that it implies that a control function generator will have to be constructed which has a cyclic process. In view of this, the control function generator shown in Fig. 26 was designed.

The operation of the control function generator will be explained in terms of what system function each output controls. The outputs of the generator are labeled 1 through 14 and are labeled in the same manner at the terminals controlled on the previous digital logic diagrams. Assuming

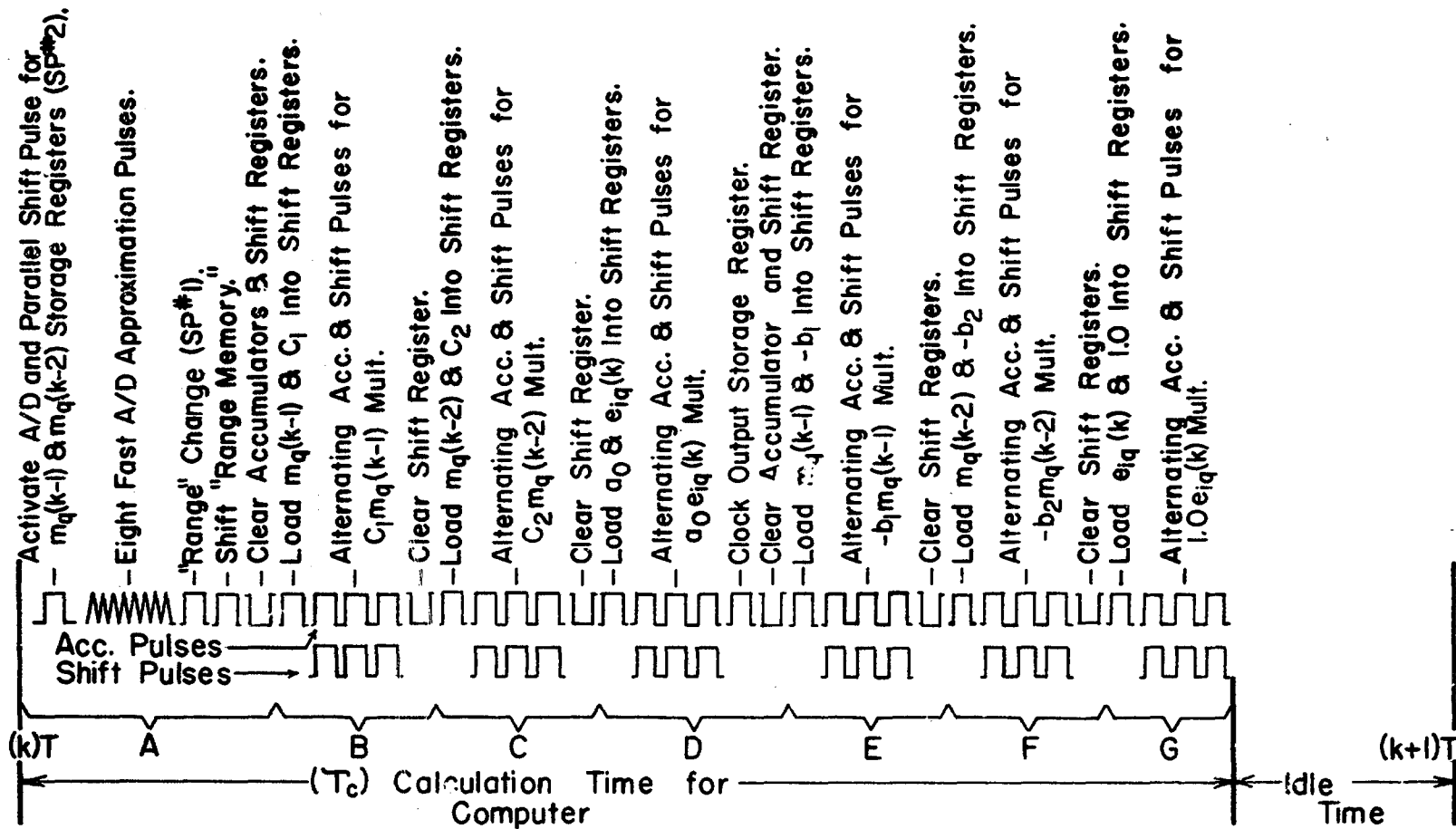
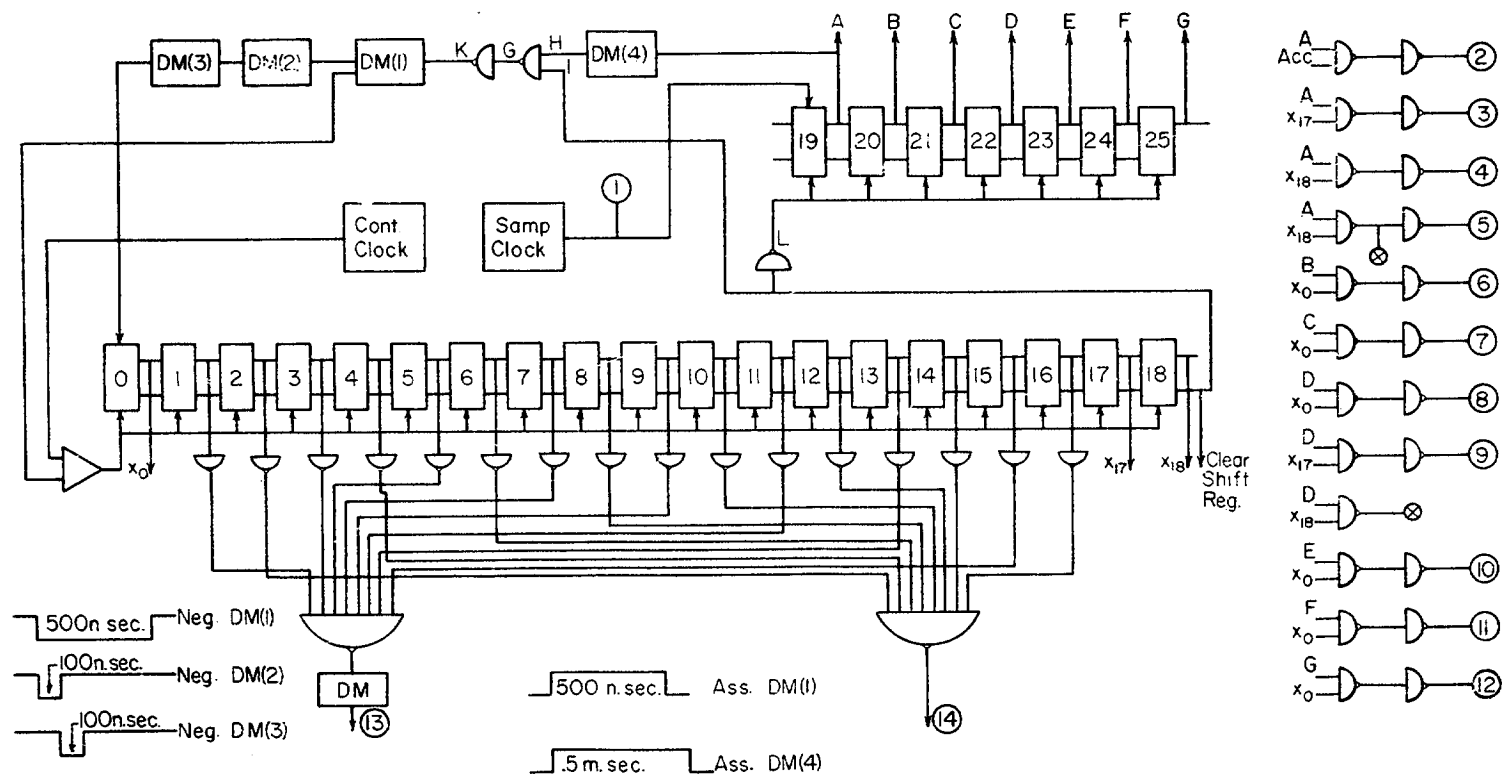


Fig. 25. Timing and pulse diagram for sequence of operation.



that all flip-flops of the generator are initially in the zero state, the sampling and continuous clocks are first turned on. On the first 1-0 transition at output 1, the A/D is activated and the $m_q(k-1)$ and $m_q(k-2)$ storage registers are shifted in parallel. This is shift pulse # 2. Since f-f's 0 through 18 were initially in the "zero" state, the continuous clock applied to their clock terminals has no effect as the f-f's are in the "zero" state. The first 0-1 transition at DC set terminal of f-f No. 19 sets this flip-flop to the "one" state. This causes a 0-1 transition at the input of DM 4 which triggers it and a .5mc assertion pulse results at the output. This pulse remains at the logic "one" level throughout the complete calculation time of the computer. As point I is in the "one" state, the 0-1 transition at point H causes a 0-1 transition at the input of DM 1. It's negation pulse disables the continuous clock and while it is disabled f-f 0 is set to the "one" state by the assertion output of DM 3. The assertion pulse of DM 1 was used to trigger DM 2 which in turn triggers DM 3. DM 2 is used to make sure that the continuous clock has been disabled before f-f 0 is set to the "one" state. If the continuous clock were not disabled while f-f 0 is being set to the "one" state, several f-f's might be set to the "one" state instead of the only desired one. Now that f-f 0 has been set to the "one" state, the continuous clock is enabled and the "one" is clocked through the first eighteen flip-flops. By doing this the eight fast A/D approximation pulses are provided, the range is changed if needed, the range memory is shifted, the accumulator is cleared, and finally the shift registers are cleared. The digital logic needed to produce the

control pulses required to do the above is shown in Fig. 26 also. The DM's at certain points in Fig. 26 are used so that sharper and narrower pulses can be formed.

As the logic "one" was shifted into and out of f-f 18, a 0-1-0 pulse was generated at point L which shifted the logic "one" out of f-f 19 and into 20 and the 1-0-1 transition at the reset output of f-f 18 triggers DM 1 which in turn by the process described earlier sets f-f 0 to the "one" state. Now point B is at logic level "one" and as the logic "one" is clocked through the first 18 f-f's, the $c_1 m_q(k-1)$ multiplication process is accomplished. Likewise points C and D in turn go to logic "one" levels and along with the process of shifting the logic "one" through the first eighteen flip-flops, $e_{oq}(k)$ is calculated and stored so that it can be changed to an analog voltage level. Finally the accumulator is cleared. Next $m'(k)$ is calculated as points E, F, and G go to logic level "one" and the logic "one" is successively clocked through f-f's 0 through 18. At this point, the computer has completed its operation for this particular sampling period. The time taken to complete the above operations is τ_c on the timing diagram. Since τ_c is less than the sampling period, the computer remains idle until the next sampling period arrives, thus accounting for the idle time shown in the timing diagram.

It might be wondered how the frequencies of the sampling and continuous clocks were determined. The PIGA loop and the Saturn V t.v.c. system both require that the input signal $e_1(t)$ be sampled eight times a second, thus determining the speed of the sampling clock. The

continuous clock speed is determined in a different manner. It is desired to run this clock as fast as possible so that τ_c can be reduced. If τ_c is kept small, then if ever desired, the sampling clock can be run faster than 8 Hz. In determining the speed of the continuous clock, it is known that for satisfactory operation of the compensator, the accumulate and shift pulses must be a minimum of 3.5 μ s apart. Thus from Fig. 26 for a practical example, it must be a minimum of 7 μ s between the time a logic "one" enters f-f 12 and the time it enters f-f 14. Knowing this, the speed of the continuous clock can be calculated to be 285 KHz.

Data transfer logic.

As stated earlier, a method must be provided by which the coefficients and the internal variables can be transferred to their respective shift registers when needed for a multiplication process. The information entered into the shift registers is entered in parallel through the DC set terminals of the flip-flops. Since a flip-flop is set to the "one" state when its DC set terminal remains at logic "zero" for 50 ns or longer, the data transfer logic will have to be designed to do this on command from the control function generator when a logic "one" is desired in the f-f. On the contrary, the data transfer logic will also have to be able to load a logic "zero" into certain f-f's on command from the control function generator. For the later case, as will be seen later, this is done by not applying a pulse to the DC set terminals of the f-f's that are to be set to the "zero" state since they were set to this state when the shift registers were cleared. The logic diagrams of the data transfer logic which will perform as described

above are shown in Figs. 27 and 28.

In explanation of how the data transfer logic works, one may assume that the coefficient c_1 is to be put into the coefficient shift register. Figure 27 will be used in explaining this operation. Normally the outputs from the control function generator, 8, 6, 7, 10, 11, and 12 are at logic "zero." Thus all the outputs of the two-input nand gates are at logic "one." Now to load the c_1 coefficient, a 0-1-0 pulse is received at 6. If the sign bit of c_1 were a logic "one," the output of the two input nand-gate of this bit position undergoes a 1-0-1 transition as the 1-0-1 pulse is applied at 6. Because of the logic used, all of the nand-gates have the same logic at their outputs at all times. Now since the output of the two input nand gate underwent a 1-0-1 transition, the DC set terminal of the (SB) position of the coefficient shift register underwent the same transition, thus setting it to the one state. If the (SB) were a logic "zero" when the 0-1-0 pulse was applied to 6, then the output of the two input nand-gate would not have undergone a change, thus leaving the (SB) of the coefficient shift register a logic "zero." The other bit positions of c_1 undergo the same type operation, thus enabling the 0-1-0 transition at 6 to load c_1 into the shift register.

The same type procedure as above is used to load the internal variables into the internal variable shift register.

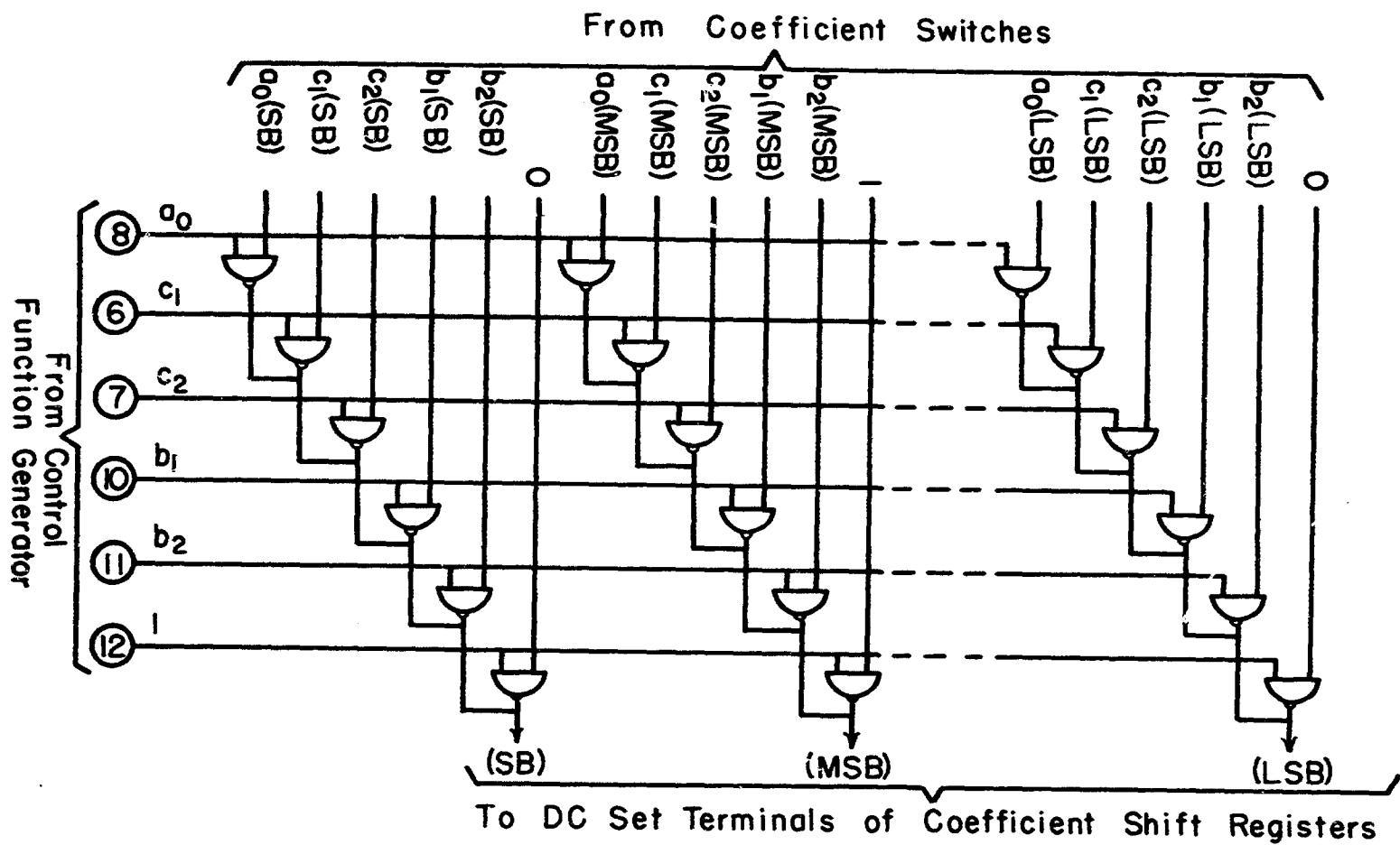


Fig. 27. Data transfer logic for coefficients.

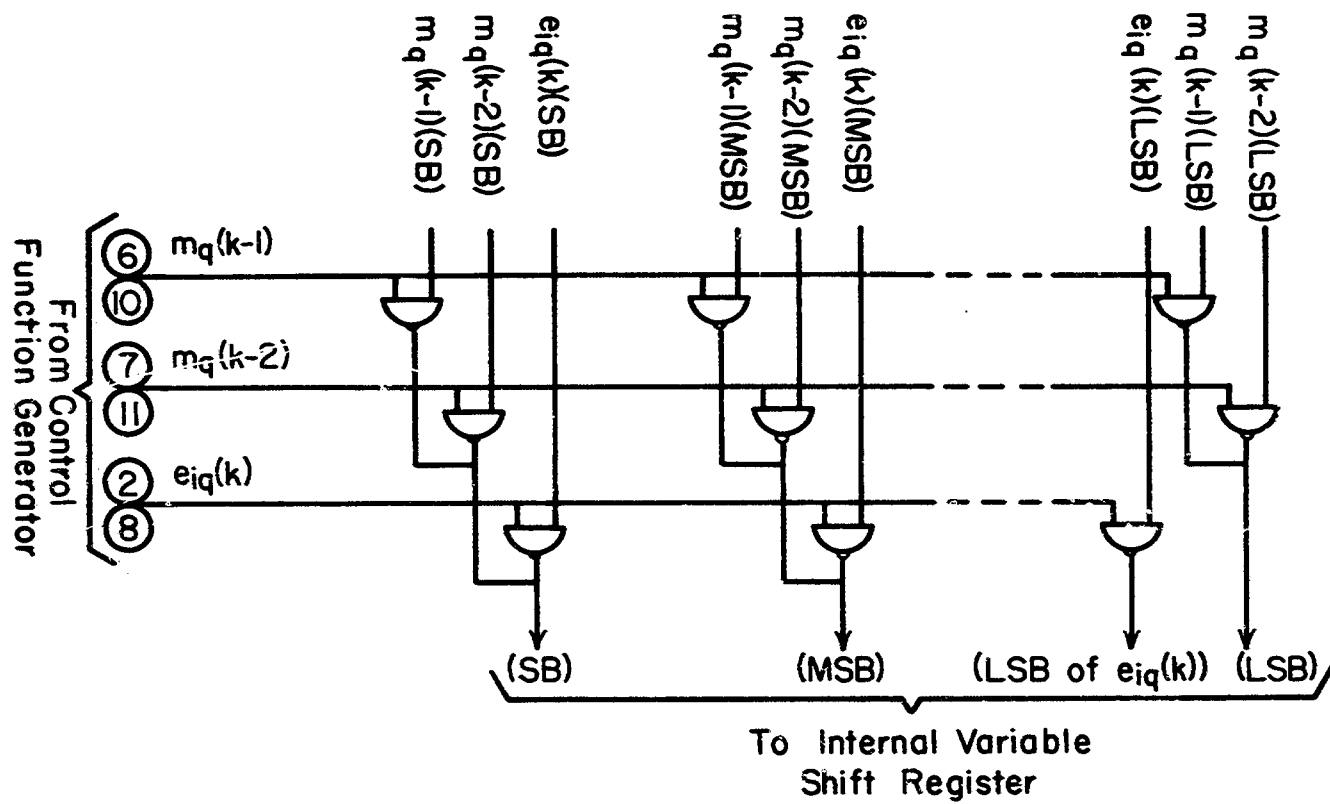


Fig. 28. Data transfer logic for internal variables.

IV. PERFORMANCE OF THE "RANGE SWITCHING" DIGITAL FILTER REALIZATION

Performance tests were made of the "range switching" digital filter when the design and the digital implementation of the filter were completed. The tests were required so that a study could be made to determine if the filter successfully stabilized the Saturn V t.v.c. system and the PIGA loop. Both open-loop and closed-loop performance tests were run. The open-loop tests were run on the compensator alone, while the closed-loop tests were run with the compensator in both of the two above mentioned control loops. The open-loop tests were run to obtain the step and sinusoidal response of the filter while the closed-loop tests were run to obtain the response of the PIGA loop to a unit step and a noise input. Also closed-loop tests were run on the Saturn V t.v.c. system to determine the performance of the filter with four different quantization schemes. The four schemes will be known as the "range switching" case, the ideal case, A/D quantization only, and finally the non range-switching ideal case. A more detailed description of each of the four just mentioned schemes is presented later in the chapter.

After the above performance tests were simulated on the IBM 7040 computer, their results were evaluated and it was decided if it would be feasible to physically construct the compensator and employ it in the actual mentioned control loops.

Open-Loop Performance

From [11] and the previous mentioned simulations run on the IBM 7040 computer in conjunction with the design of the digital filter, it was found that a $D(z)$ of

$$D(z) = \frac{1.875z^{-1} - 1.75z^{-2}}{1 - 1.125z^{-1} + .375z^{-2}} \quad (4-1)$$

successfully stabilizes the Saturn V t.v.c. system. To perform the open loop performance tests, the desired analog input was applied to the digital compensator of Fig. 6 and the quantized digital output $e_{oq}(k)$ recorded. To avoid having to perform manually the calculations required each sampling period to obtain the output, the digital filter of Fig. 6 was simulated on the IBM 7040 computer.

Step response. To obtain the step response of the digital filter, a unit step function with a magnitude of one was applied to the input of the filter shown in Fig. 6. As seen from Fig. 29, the filter has a transient output response which levels off at about .3516. The output of the filter can be expected to level off to a constant value since the input is constant. The value which the output approaches, .3516, is smaller than the final value predicted for the filter by the final value theorem of sampled-data theory. By this theorem,

$$\begin{aligned} F.V. &= \lim_{z \rightarrow 1} E_{in}(z) D(z) (z-1) \\ &= \lim_{z \rightarrow 1} \left(\frac{z}{z-1} \right) \left(\frac{1.875z^{-1} - 1.75z^{-2}}{1 - 1.125z^{-1} + .375z^{-2}} \right) (z-1) \end{aligned}$$

$$F.V. = .5,$$

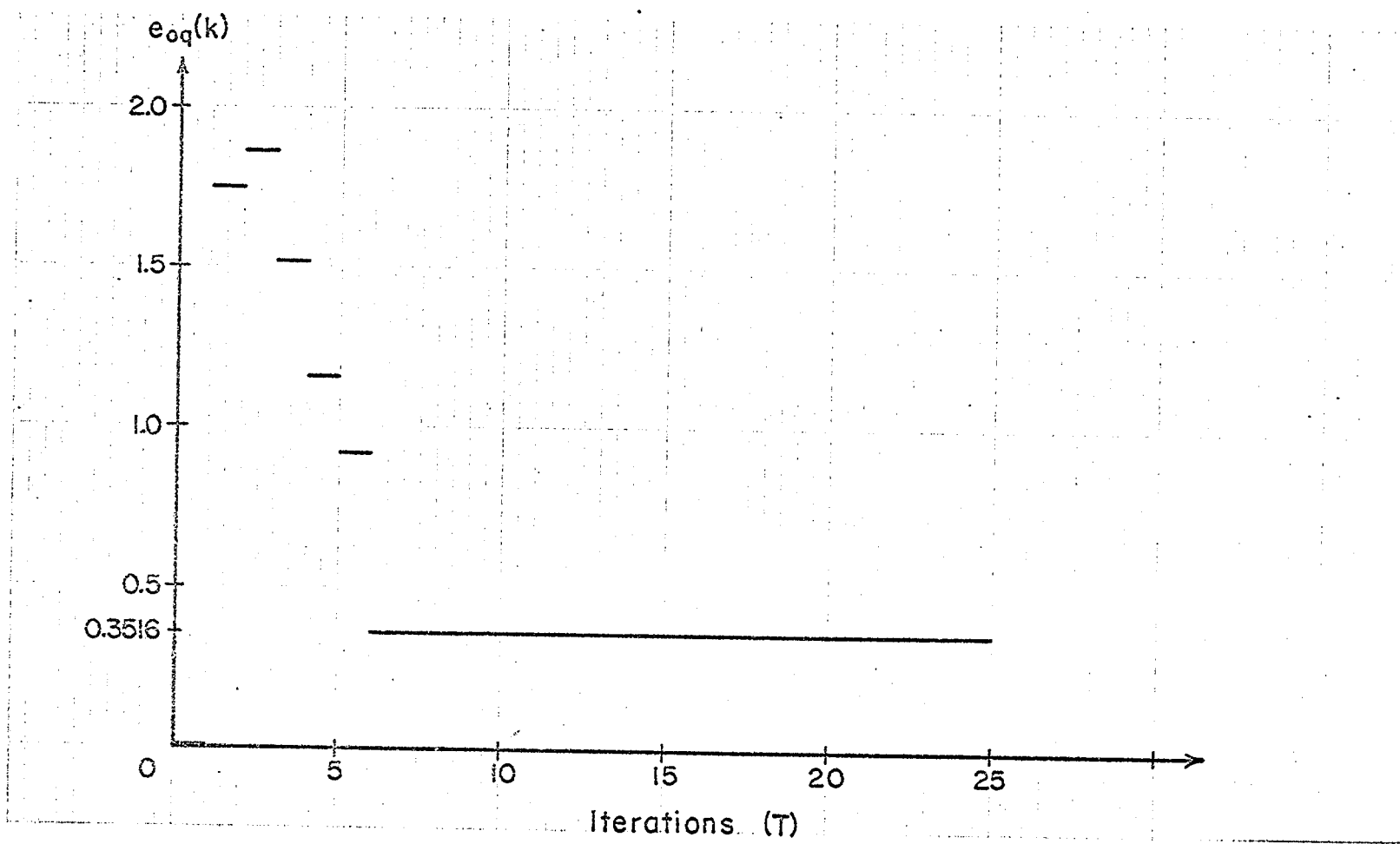


Fig. 29. Open-loop step response.

the final value should be .5. The smaller final value of the filter can be accounted for by the operation of the quantization procedures of the filter. Another characteristic of the step response noticed from Fig. 29 is that the settling time is about .6 second, which is faster than the settling time of the second order filters designed in [11] and [13]. That the output responds faster to the input of the filter is sometimes a desirable characteristic.

Sinusoidal Response. The open loop sinusoidal response of the digital filter is obtained by applying a .4 HZ. two volt peak-to-peak sinusoidal input to the filter shown in Fig. 6. To obtain the output response of the filter, a simulation of the filter with the sinusoidal input was run on the IBM 7040 computer. As can be predicted by sampled-data theory [1], the output $e_{oq}(k)$ resembles a sine wave which has been impulse sampled and fed through a zero order hold device. Two complete cycles of the steady state response are shown in Fig. 30. Looking at the figure, it is seen that the output response is regular except at the peaks. The irregularity of the peaks is due to the quantization noise of the filter [11].

Closed-Loop Performance

The closed-loop tests will now be run to help determine if the compensator suitably stabilizes the PIGA and Saturn V t.v.c. control loops. The digital filter was first simulated in the PIGA loop and then in the Saturn V t.v.c. control loop.

PIGA-loop performance. A schematic diagram of the PIGA control

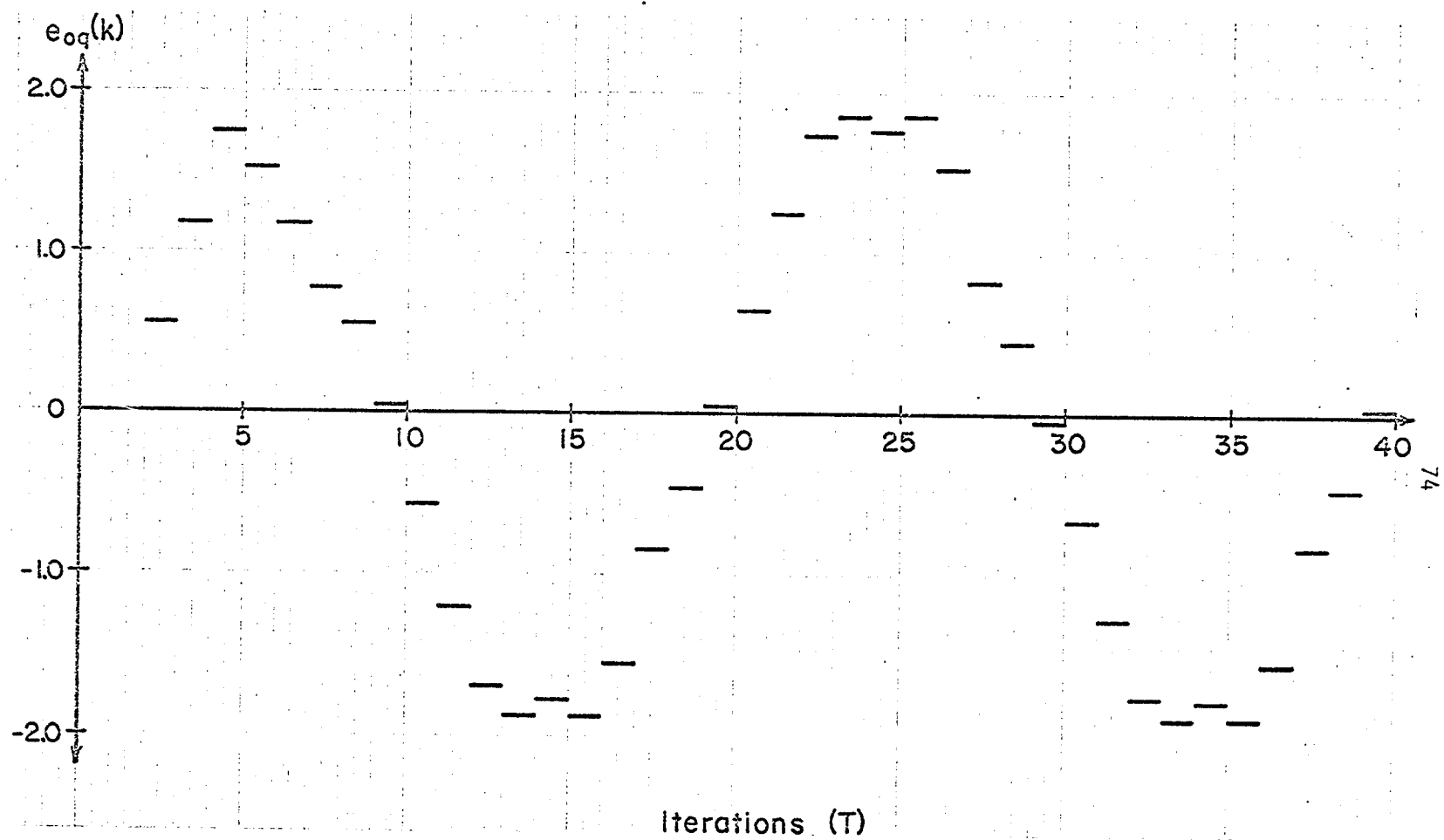


Fig. 30. Open-loop sinusoidal response.

loop along with the values of the variables of the loop is shown in Fig. 31. In explanation of each block in Fig. 31, G_p is a gain which converts the angle β into an analog voltage input for the A/D which is the input sampling device of the compensator shown in the figure. The sensing scheme of the angle β is explained in full by [15]. The compensator $D(z)$ and the zero-order hold $G_{ho}(s)$ of the figure describe mathematically the effect of the digital filter on the PIGA loop. The output gain G_T shown in the figure converts the binary output word of the digital filter to an output torque by the use of a pulse-width D/A, a PWM power amplifier, and a torque motor.

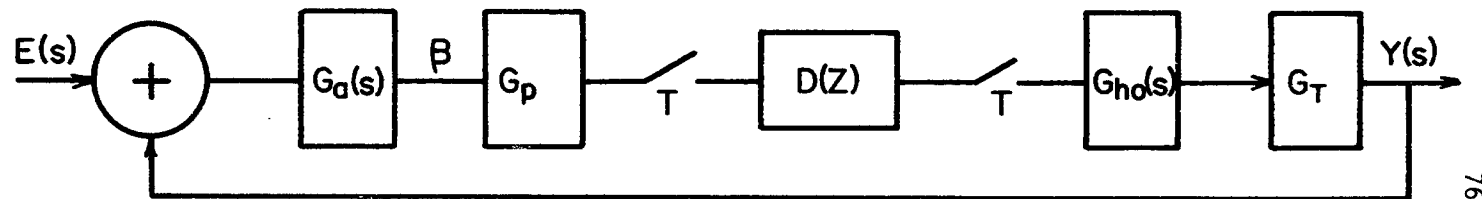
Using a $D(z)$ of the form

$$D(z) = \frac{1 - 119/64z^{-1} + 57/64z^{-2}}{1}$$

with a sampling rate $f_s = 10000$ Hz.,

a unit step input and then a random noise input will be applied to the input of the PIGA loop. The main item of interest of these two particular performance tests is the error angle β , which is desired to reach a minimum value as soon as possible after an initial disturbance is applied to the loop. In this case the disturbance will be the unit step and the random noise inputs. A plot of β vs. time for the unit step input is shown in Fig. 32 and the same plot for the noise input is shown in Fig. 33.

Looking at Fig. 32, it is seen that the compensator successfully stabilizes the PIGA loop with a unit step function input. From the



76

$$G_a(s) = (\omega_n^2/H)/s(s^2 + \omega_n^2)$$

T = sampling period

$$G_p = 1.0 \text{ volt-peak/deg.}$$

$$G_{ho}(s) = (1 - e^{-Ts})/s$$

$$G_T = 5.46 \text{ gm-cm/binary level}$$

$$\omega_n = 184 \text{ rad/sec.}$$

$$H = 10^5 \text{ gm-cm}^2/\text{sec.}$$

Fig. 31. Block diagram of PIGA control loop.

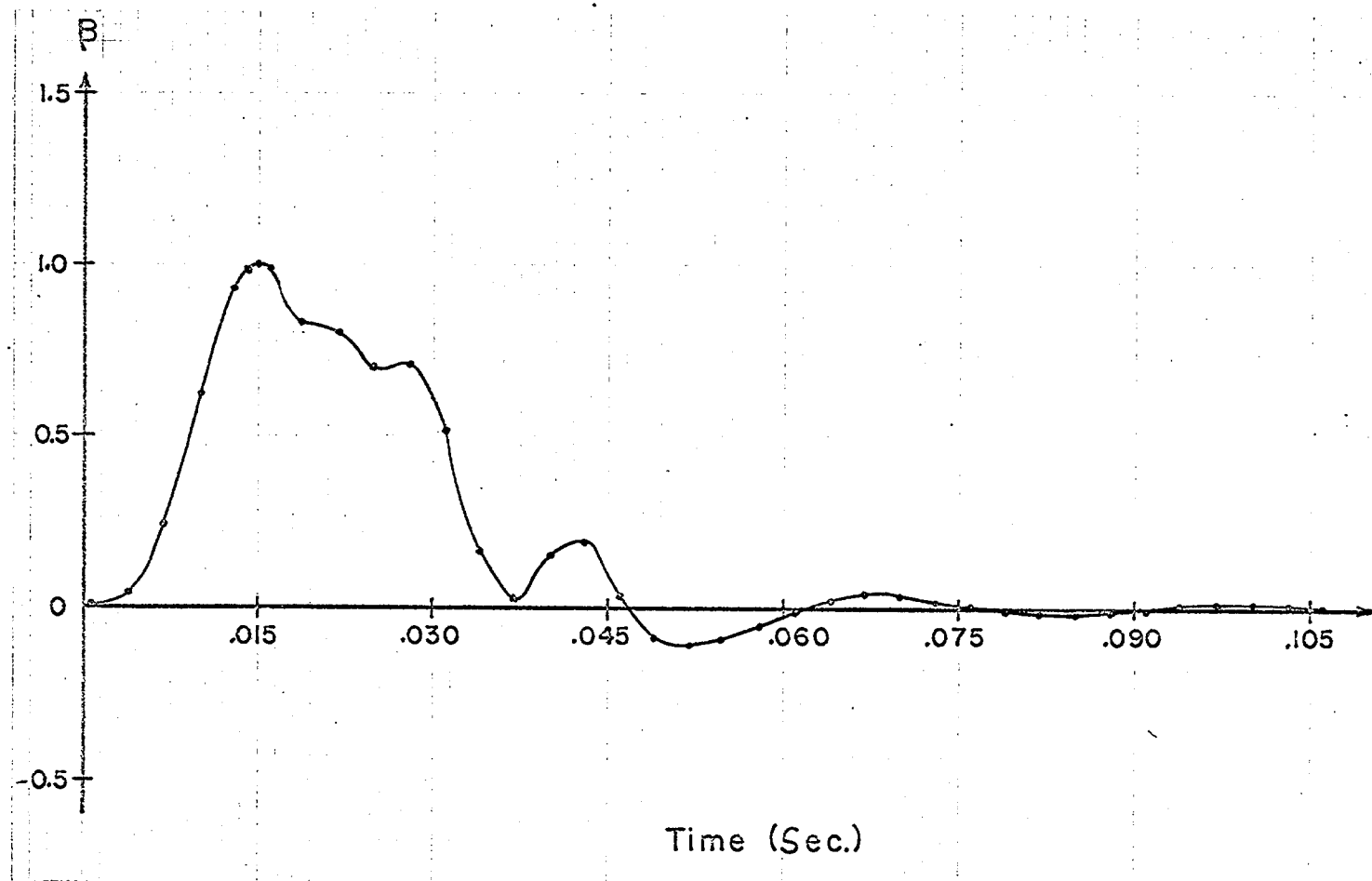


Fig. 32. Response of β to a unit step input.

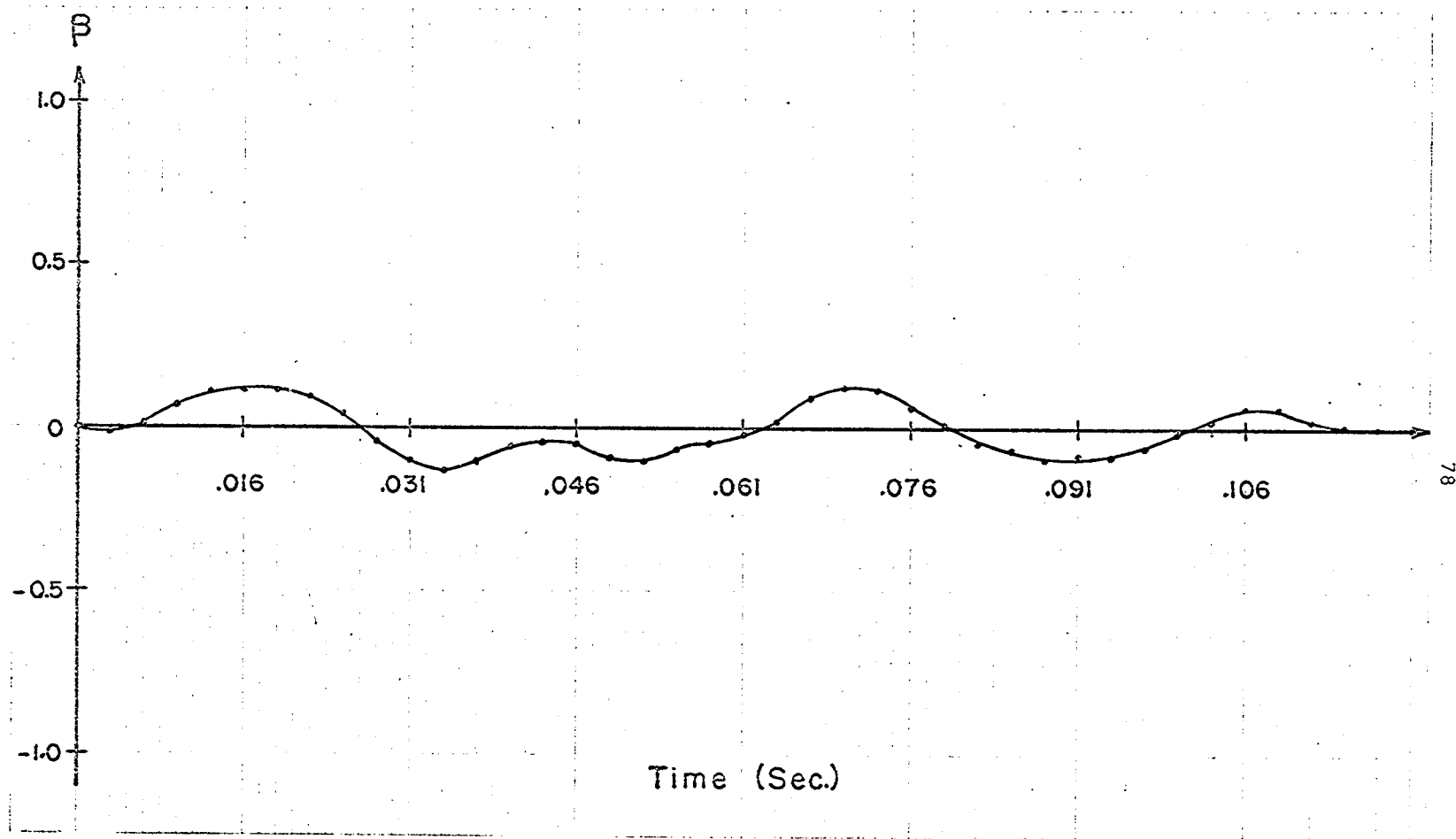


Fig. 33. Response of β to a noise input.

figure, β goes up to a maximum value of one degree and then settles down to an average value of 0 degrees. The irregularity of β between .031 seconds and .046 seconds is caused by the switching of the compensator from the "higher range" of operation to the "lower range." It appears that the compensator momentarily discontinues control of the system when the range of operation is switched, but this has no adverse effects since control is quickly regained.

The random noise input into the PIGA loop was applied so that it could be seen if the system would remain stable when the input varies at random as it would in a physical situation. From Fig. 33, it is seen that the system does remain stable and that the error angle β tends to oscillate with an average value of 0 degrees over a period of time. The magnitude of the oscillating error angle β is within the specifications required for successful operation of the PIGA loop.

Saturn V t.v.c. system performance. The Saturn V t.v.c. system performance tests were run to determine the ability of this system, which contains the previously designed $D(z)$, to return the rocket to a pre-assigned flight attitude path after experiencing a disturbance which makes it veer off course 10 degrees. The main point of interest is how fast the rocket is returned to its original flight attitude path after a disturbance. To make this study the flight of the Saturn V, with an initial pitch error angle ϕ_d of 10 degrees, was simulated on the IBM 7040 computer. A plot of the system response to the initial error, ϕ_d vs. time, can be drawn from the program output.

Four different simulations of the rockets flight were run, the

main difference in each one being the amount of quantization used. This was done so the effects of the quantization at different points could be observed. For each simulation, the compensator had a sampling rate of eight cycles-per-second and realized a $D(z)$ in the form of equation (4-1).

For the first simulation, the compensator used in the t.v.c. system was the same one as designed in the previous two chapters. It contained quantization at the three preciously mentioned points. The plot of the system response to an initial pitch error angle of 10 degrees is shown in Fig. 34. From the figure it is seen that it takes about 20 seconds for the attitude of the rocket to be corrected and after this it oscillates with a maximum magnitude of $\pm .20$ degree, which is within the system specifications.

The second simulation run was the ideal case, i.e., all quantization and the "range switching" process of the designed compensator were removed. From the system response shown in Fig. 35, it is seen that the first overshoot is less than the same overshoot experienced by the "range switching" compensator of the previous simulation and the system settles down slightly faster and also to a lower value of about $\pm .01$ degree.

For the third simulation, the compensator contained quantization at the A/D only and the range switching process was removed. Again, like the simulation in which there was not any quantization at all, the first overshoot was a little less than for the range switching simulation as can be seen from Fig. 36. Also the error angle ϕ_d levels

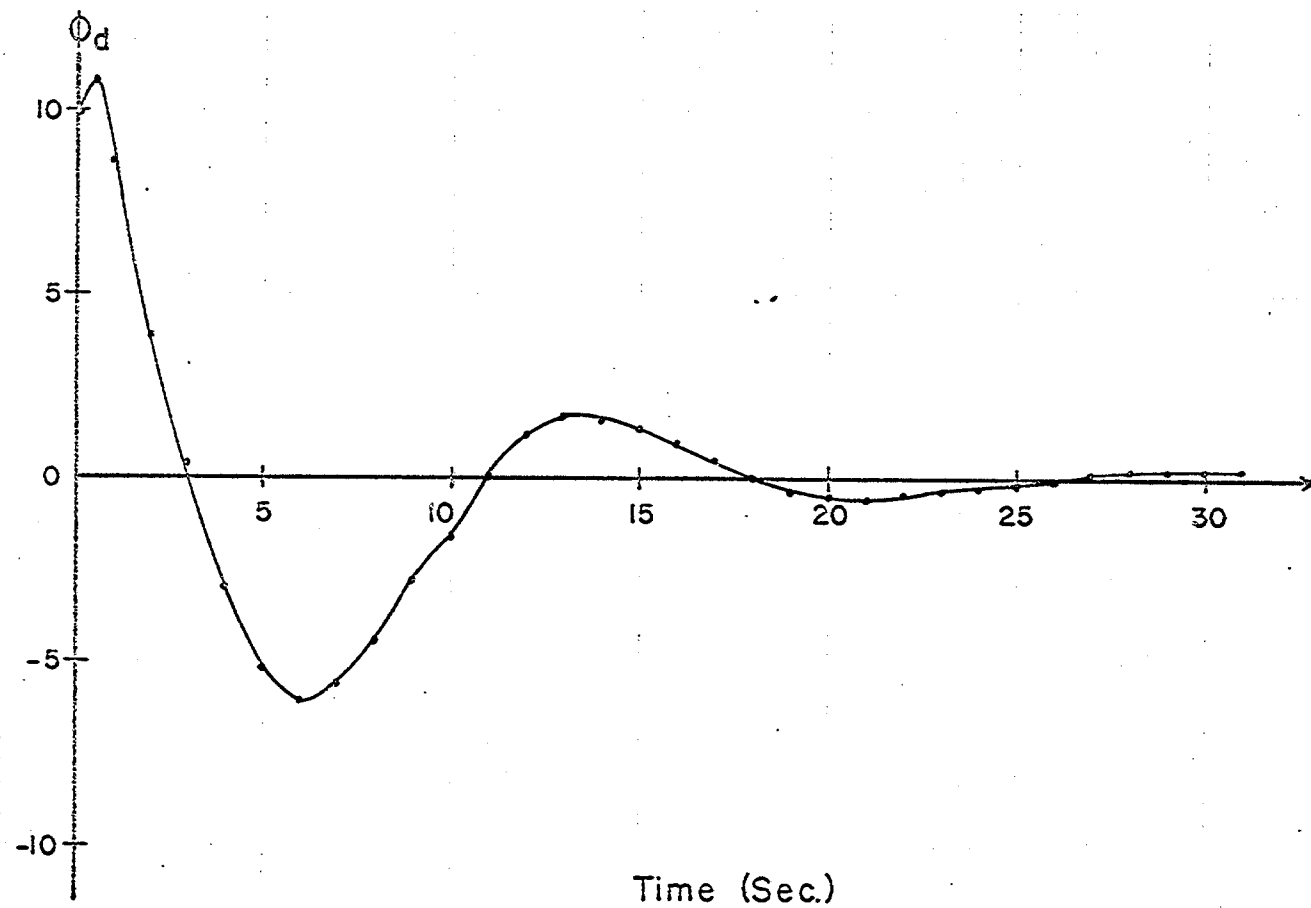


Fig. 34. Response of "range switching" filter.

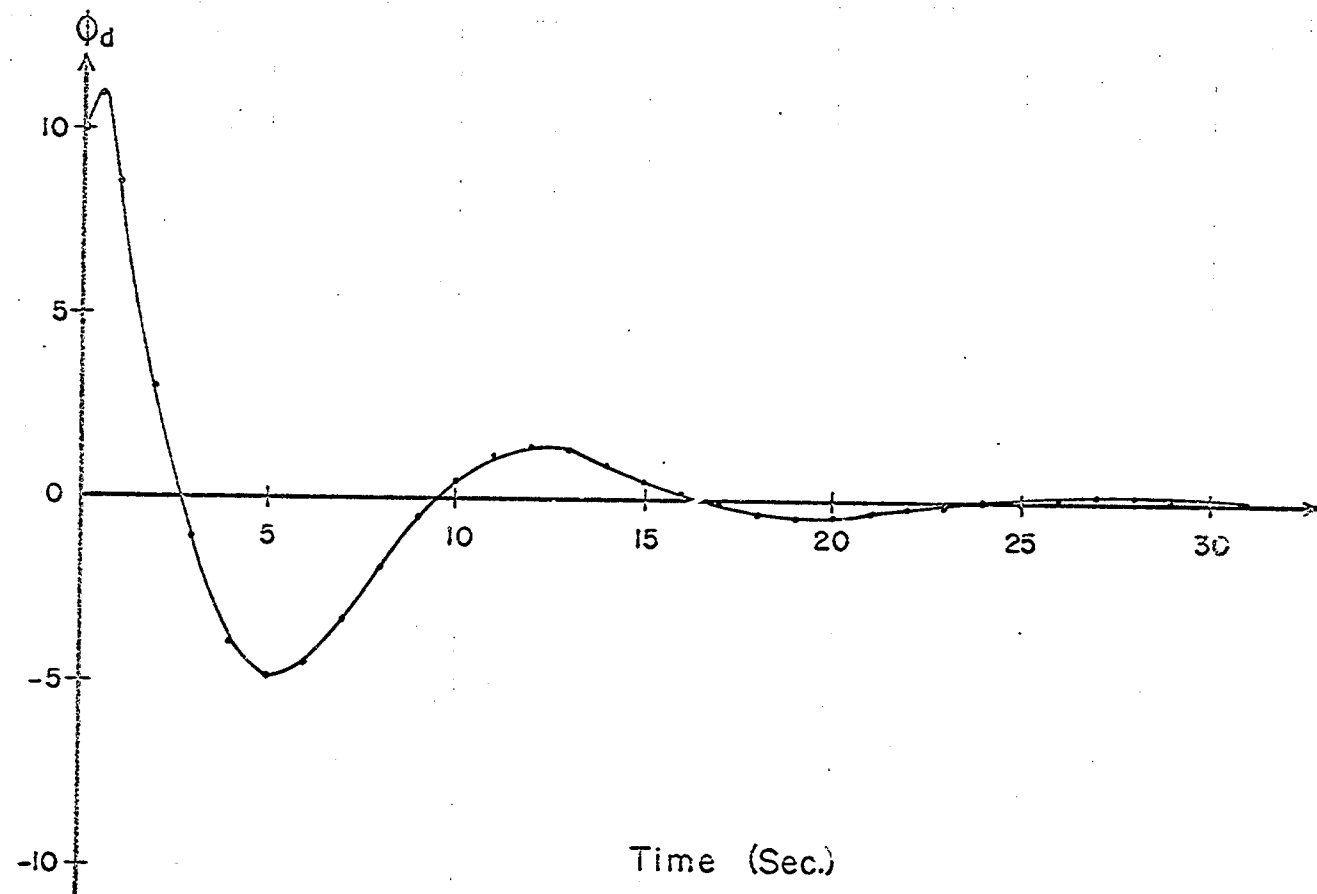


Fig. 35. Response of ideal case.

off to about 0.1 degree which is within system specifications but greater than the ideal case.

Last, a simulation was run in which the compensator was the same as in the first simulation except the "range switching" process, which weights the internal variables properly each sampling period, was omitted. The compensator operating in this manner will be known as the "non-range switching" ideal filter. The response of the system with this type compensator is shown in Fig. 37. From the figure, it is seen that it has about the same identical response as the "range switching" version of the first simulation.

In summary of the chapter, it is seen from the PIGA and Saturn V t.v.c. system performance tests that the preciously designed digital filter successfully stabilizes both systems. In choosing one of the four types of filters employed in the performance tests to stabilize the Saturn V t.v.c. system, one must consider the accuracy desired, the range of sampling frequencies the compensator is able to operate over, and the physical size of the compensator. In considering the above three characteristics, it is believed that the "range switching" digital filter of the first simulation is best suited to meet these specifications. It has a wide range of sampling frequencies, varying from 0 Hz. to 2000 Hz., reasonable accuracy, and most important where space and physical size is a limitation; it has a physical size of about two-thirds of that of a "non-range switching" digital filter.

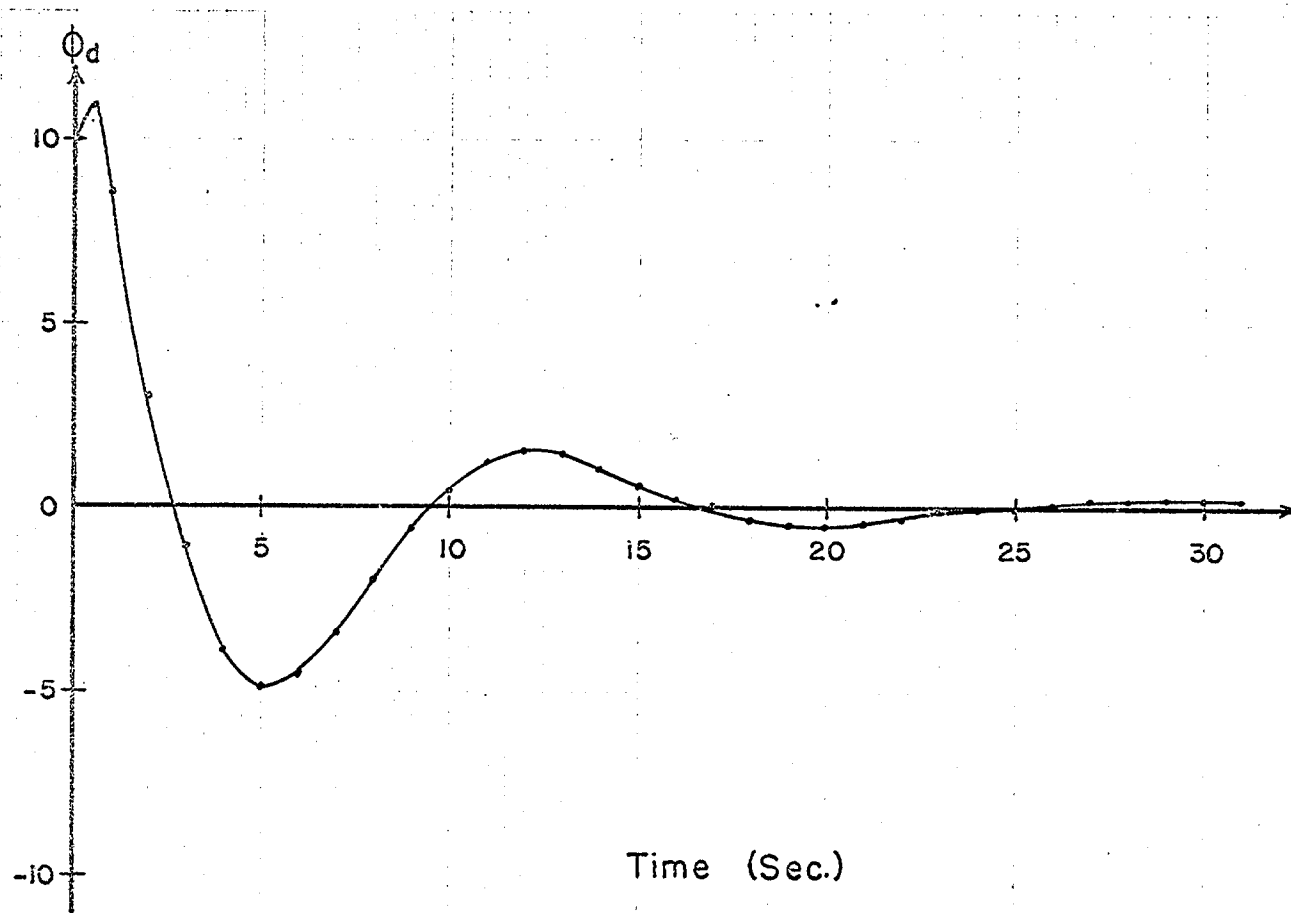


Fig. 36. Response with A/D quantization only.

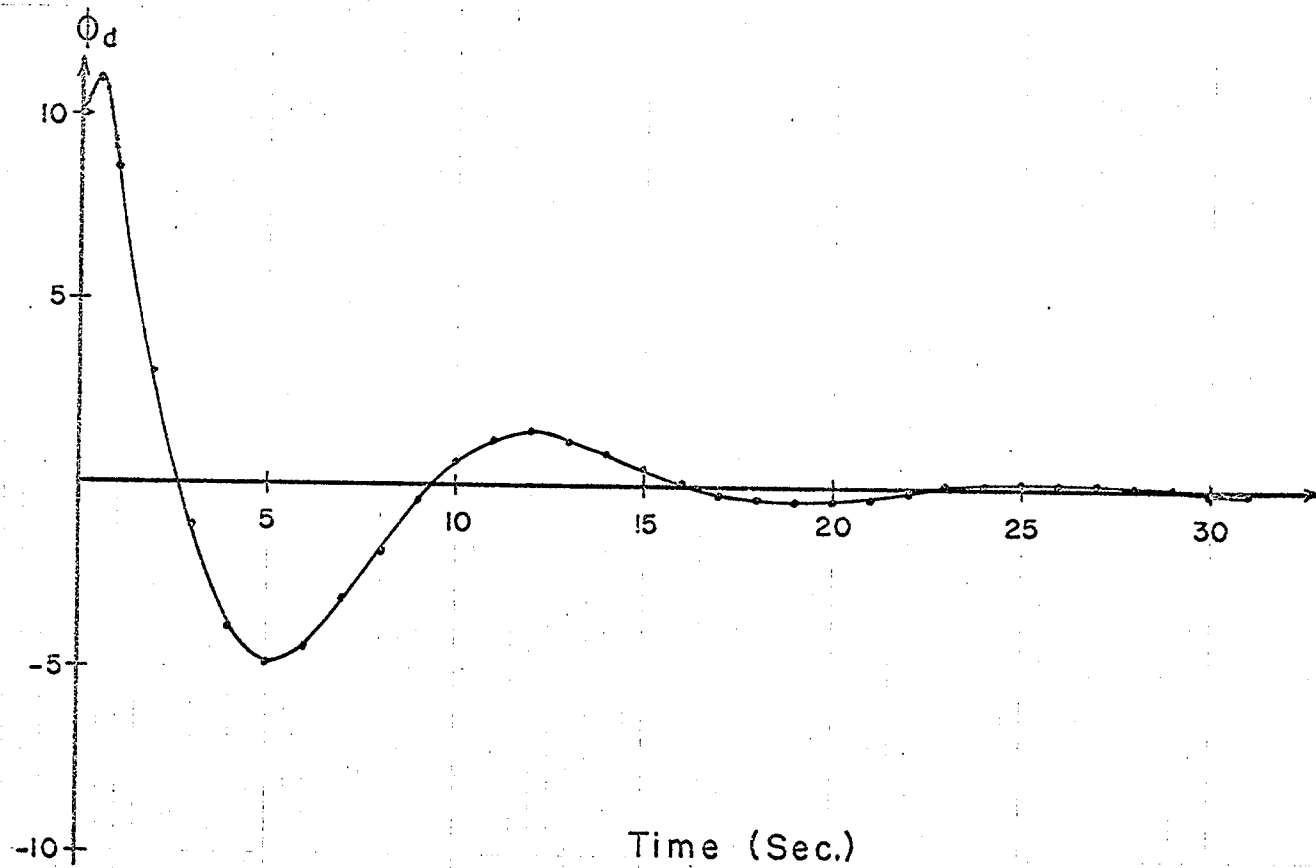


Fig. 37. Response of "non-range switching" ideal filter.

V. CONCLUSIONS

This paper has presented the design and a possible digital implementation of a "range-switching" digital filter which may be used in various sampled-data control systems.

The first step in the design of the filter was the selection of a suitable mathematical programming form which would require a minimum of hardware to physically construct the filter. Also of importance: the mathematical programming form chosen was one that required a minimum calculation time, thus enabling the filter to have a wider range of sampling frequencies. Next, the "range switching" process was mathematically defined and its effects on the internal variables of the filter noted so that a digital implementation of the process could be accomplished. Finally, a digital implementation of each component of the filter was proposed and then open and closed-loop performance tests were run on the filter.

The filter's performance in stabilizing the two control loops it was designed for, the PIGA and Saturn V t.v.c. system, was very satisfactory. It demonstrated a quick response to disturbances and was able to gain fine-granularity control over both loops. As stated earlier, the filter has a relatively wide sampling frequency range which is advantageous in that it allows the filter to be compatible in a wider variety of control loops. Considering other characteristics of the filter discussed in

Chapter IV and the above mentioned ones, it is believed that the use of "range switching" digital filters similar to the one designed in this paper will have wide-spread effects in future digital control.

REFERENCES

- [1] B. C. Kuo, Analysis and Synthesis of Sampled-Data Control Systems. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1963.
- [2] H. H. Hull, C. C. Carroll, and H. T. Nagle, Jr., "On the realization of a generalized second-order digital compensator," Record of the IEEE 1968 Region 3 Convention, pp. 13.1.1-13.1.5, April, 1968.
- [3] H. T. Nagle, Jr., and C. C. Carroll, "Organizing a special-purpose computer to realize digital filters for sampled-data systems," IEEE Trans. on Audio and Electroacoustics (Special Issue on Digital Filters), Vol. AU-16, No. 3, (in print), September, 1968.
- [4] R. W. Kearns, "A digital compensator for automatic control systems," Grant NsG-36-30, Case Institute of Technology, Cleveland, Ohio, June, 1964.
- [5] D. J. Gawlowicz, "A programmable digital compensator for single-loop high performance digital servomechanisms," Grant NsG-36-20, Case Institute of Technology, Cleveland, Ohio, 1965.
- [6] Y. Cho and F. G. Popp, "A high-speed arithmetic unit for digital filter implementation," NEREM-67, Boston, Massachusetts, November, 1967.
- [7] C. C. Carroll and R. White, "Discrete compensation of control systems with integrated circuits," IEEE Transactions on Automatic Control, Vol AC-12, No. 5, pp. 579-582, October, 1967.
- [8] H. T. Nagle, Jr., C. C. Carroll, and J. W. Jones, Jr., "A Hybrid analog-digital controller for realizing sampled-data control functions," Proc. of the Second Annual Princeton Conference on Information Sciences and Systems, (abstract) p. 256, March, 1968.
- [9] J. E. Bertram, "The effect of quantization in sampled feedback systems," Transactions AIEE on Applications and Industry, pt. 2, Vol 77, September, 1958, pp. 177-181.

- [10] R. K. Cavin, C. L. Phillips, and D. L. Chenoweth, "A least upper bound on quantization error," IEEE Transactions on Automatic Control, Vol. AC-13, No. 3, June, 1968.
- [11] C. C. Carroll, et al., "The Hybrid Realization of a Digital Controller for the PIGA Control Loop," Technical Report No. 6, NAS8-20163, Auburn Research Foundation, Auburn, Alabama, September, 1967.
- [12] L. B. Jackson, J. K. Kaiser, and H. S. McDonald, "Implementation of Digital Filters," IEEE International Convention-68, New York, N. Y., March 1968.
- [13] C. C. Carroll, et al., "On the Realization of a Generalized Second-Order Digital Compensator," Technical Report Number 9, NAS8-11274, Auburn Research Foundation, Auburn Research Foundation, Auburn, Alabama, September, 1967.
- [14] A. Deerfield, "Canonical Digital Filters," NEREM-67, Boston, Massachusetts, November, 1967.
- [15] C. C. Carroll, et al., "Interfacing Digital Compensators into the PIGA Loop," Technical Report Number 8, NAS8-20163, Auburn Research Foundation, Auburn, Alabama, March, 1968.
- [16] C. L. Phillips, et al., "Digital Compensation of the Thrust Vector Control System," Technical Report Number 8, NAS8-11274, Auburn Research Foundation, Auburn, Alabama, May, 1967.
- [17] C. L. Phillips, et al., "Digital Compensation of the Thrust Vector Control System," Technical Report Number 7, NAS8-11274, Auburn Research Foundation, Auburn, Alabama, January, 1967.
- [18] C. L. Phillips, et al., "Digital Compensation of the Thrust Vector Control System," Technical Report Number 10, NAS8-11274, Auburn Research Foundation, Auburn, Alabama, January, 1968.
- [19] C. C. Carroll, et al., "A Special-Purpose Computer Realization of a Third-Order Digital Filter for the PIGA Control Loop," Technical Report Number 9, NAS8-20163, Auburn Research Foundation, Auburn, Alabama, May, 1968.
- [20] Y. Chu, Digital Computer Design Fundamentals, New York, N. Y., McGraw-Hill Book Co., Inc. 1962.

APPENDIX A

FORTRAN SOURCE PROGRAM FOR SIMULATION OF FIGA LOOP WITH A NOISE INPUT

SOURCE DECK

```

$JOBW EED55T JRHEATH
C    TIME DOMAIN SIMULATION OF THE COMPENSATED SYSTEM
      DIMENSION X1(2),X2(2),X3(2),      D(2)
      COMMON/COM2/RM(1001), ITER, TES
      COMMON/COM1/XP(3,2), BQ(1), C(2)
      1 FORMAT(1H ,13,2X,1P9E13.4)
      CALL INPUT
      2 FORMAT(1H1)
      N=1
      H=1.0E+05
      WN=184.0
      T=0.001
      GP=56.2
      GT=321000.0
C    GDA IS THE TOTAL LOOP GAIN
      GDA=0.083
      W1=SIN(WN*T)/WN
      W2=COS(WN*T)
      W3=(1.0-W2)/WN**2
      W4=(T-W1)/H
      W5=(1.0-W2)/H
      W6=WN*SIN(WN*T)
      W7=W6/H
      GDIG=GDA/GT/GP/(T-W1)*H
      WRITE(6,10) GDIG
      10 FORMAT(1H0,7HKDIG = ,1PE11.4)
      TEST=+2000.0*980.0/4.0
      DO 669 NAGL=1,2
669   XP(NAGL,1)=0.0
      8 BDC=0.0
      BQ(1)=0.0
      X1(1)=0.0
      X2(1)=0.0
      X3(1)=0.0
      D(1)=0.0
      C(1)=0.0
      COFS=0.0
      DO 5 1=1, ITER

```

```

R=TES*980.*RM(I)
4  WRITE(6,1) I,BDC,BQ(N),X1(N),XP(1,1),D(N),R,COFS,C(1)
C  BEGIN ANALOG PORTION SIMULATION
    X1(N+1)=X1(N)+W1*X2(N)+W3*X3(N)+W4*(R-GT*D(N))
    X2(N+1)=W2*X2(N)+W1*X3(N)+W5*(R-GT*D(N))
    X3(N+1)=W2*X3(N)-W6*X2(N)+W7*(R-GT*D(N))
    BDC=GP*X1(N+1)
C  END ANALOG PORTION SIMULATION
C  BEGIN DIGITAL UNIT SIMULATION
    UI=BDC
    CALL DIGCOM (UI,YP)
    D(N+1)=GDIG*YP
C  END DIGITAL UNIT SIMULATION
    COFS=GT*D(N+1)
    X1(N)=X1(N+1)
    X2(N)=X2(N+1)
    X3(N)=X3(N+1)
5  D(N)=D(N+1)
    STOP
    END
$IBFTC DIGCOM
    SUBROUTINE DIGCOM(UI,YP)
    COMMON/COM1/XP(3,2),BQ(1),C(2)
    AO=1.0
    A1=-119./64.
    A2=57./64.
    B1=0.0
    B2=0.0
    FX=256.
    UI=UI/3.0*FX
    EX=1.0
    CALL ROUND (UI,EX,FX)
    UP=UI
Z  BQ(1)=UP*3.0/FX
    UI=UI*3.0/FX
    IF(ABS(UP)-16.0) 2,3,3
2  C(2)=0.0
    GO TO 4
3  C(2)=1.0
    UP=UP/16.
    IUP=UP
    UP=IUP
    FX=16.
4  IF(C(2)-C(1)) 5,6,7
5  XP(1,1)=16.*XP(1,1)
    XP(2,1)=16.*XP(2,1)
    AX=4.0
    BX=63.75
    CALL ROUND(XP(1,1),AX,BX)
    CALL ROUND(XP(2,1),AX,BX)
    GO TO 6
7  XP(1,1)=XP(1,1)/16.0

```

```

XP(2,1)=XP(2,1)/16.
AX=4.0
BX=63.75
CALL ROUND (XP(1,1),AX,BX)
CALL ROUND (XP(2,1),AX,BX)
6  XP(1,2)=-B1*XP(1,1)-B2*XP(2,1)+UP
   XP(2,2)=XP(1,1)
   AX=4.0
   BX=63.75
   CALL ROUND(XP(1,2),AX,BX)
   YP=(A1-A0*B1)*XP(1,1)+(A2-A0*B2)*XP(2,1)
1   +A0*UP
   CX=64.
   DX=255./64.
   CALL ROUND(YP,CX,DX)
   DO 1 I=1,2
1  XP(I,1)=XP(I,2)
   C(1)=C(2)
   YP=YP/FX*3.0
   RETURN
   END

```

\$IBFTC INPUT

SUBROUTINE INPUT

C

C

RANDOM INPUT

COMMON/COM2/RM(1001),ITER,TES

TES=200.

ITER=301

NRANB=6

CALL RANBIT(NRANB)

CALL RCON1(35187269)

RMAX=(2.**NRANB-1.)/2.

DO 19 I=1,ITER

RM(I)=IRAN(5)

19 RM(I)=(RM(I)-RMAX)/RMAX

RETURN

END

\$IBFTC ROUND

SUBROUTINE ROUND (A,AN,BN)

X=ABS(A)

S=A/X

IX=X*AN

XQ=IX

XQ=XQ/AN

IF(XQ-BN) 1,2,2

1 A=S*XQ

RETURN

2 A=S*BN

RETURN

END

\$ENTRY

\$IBSYS

APPENDIX B

FORTRAN SOURCE PROGRAM FOR SIMULATION OF SATURN V THRUST
VECTOR CONTROL SYSTEM EMPLOYING THE RANGE SWITCHING DIGITAL
COMPENSATOR.

SOURCE DECK

```

$IBFTC MAIN
      NN=100
      DIMENSION YPB(4)
      DIMENSION WB(4),ZETA(4),GM(4),YB(4),YBP(4),YPD(4),A(14,14)
      DIMENSION BD(14),X(14,2),PHIT(14,14),PHITI(14,14),W(14,14)
      DIMENSION Z(14,14)
      COMMON/COM1/XP(3,2),CX(2)
      CALL TRAP
      DO 699 INAG=1,3
699   XP(INAG,1)=0.0
      GC=.5
      CX(1)=0.0
11    FORMAT(4(5X,E15.8))
10    FORMAT(3(5X,E15.8))
50    CONTINUE
      READ(5,10)ALO,AL1,BO
      READ(5,10)B1,B2,B3
      READ(5,10)B4,B5,B6
      READ(5,11)B7,C1,C2,FC
      READ(5,11)ALCG, AIXX, AIE, TIME
      READ(5,11)ASE, AK3, AK4, P
      READ(5,11)AK7, AM, RP, V
      READ(5,11)WB(1), WB(2), WB(3), WB(4)
      READ(5,11)ZETA(1), ZETA(2), ZETA(3), ZETA(4)
      READ(5,11)GM(1), GM(2), GM(3), GM(4)
      READ(5,11)YB(1), YB(2), YB(3), YB(4)
      READ(5,11)YPB(1), YPB(2), YPB(3), YPB(4)
      READ(5,11)YPD(1), YPD(2), YPD(3), YPD(4)
      DO 9 I=1,4
9     WB(I)=WB(I)*2.0*3.1415927
33    FORMAT(1H1,4X,10HINPUT DATA,/)
      AK3 = AK3*57.29578
      AK4 = AK4*57.29578
      AK7 = AK7*57.29578
      PRINT 33
      PRINT 10, ALO, AL1, BO
      PRINT 10, B1, B2, B3

```

```

PRINT 10, B4, B5, B6
PRINT 11, B7, C1, C2, FC
PRINT 11, ALCG, AIXX, AIE, TIME
PRINT 11, ASE, AK3, AK4, P
PRINT 11, AK7, AM, RP, V
PRINT 11, WB(1), WB(2), WB(3), WB(4)
PRINT 11, ZETA(1), ZETA(2), ZETA(3), ZETA(4)
PRINT 11, GM(1), GM(2), GM(3), GM(4)
PRINT 11, YB(1), YB(2), YB(3), YB(4)
PRINT 11, YPB(1), YPB(2), YPB(3), YPB(4)
PRINT 11, YPD(1), YPD(2), YPD(3), YPD(4)
ITER=50
PRINT 17, ITER
17 FORMAT(1H1,2HN=13,/)
PRINT 26, TIME
26 FORMAT(15X,12HFLIGHT TIME=F5.1,1X,7HSECONDS,/)
N=14
T=1./8.
C
C  DEFINE ELEMENTS OF A MATRIX
DO 1 I=1,N
DO 1 J=1,N
1 A(I,J)=0.0
W1=34.48
ZETA1=0.434
W2=84.09
ZETA2=0.594
A(1,1)=-2.*ZETA1*W1
A(1,2)=-W1**2
DO 2 I=2,N,2
I1=I-1
2 A(I,I1)=1.0
A(3,2)=W1**2
A(3,3)=-2.*ZETA2*W2
A(3,4)=-W2**2
A(5,2)=(ALCG*ASE+AIE)*W2**2*(-W1**2)/AIXX
A(5,3)=(ALCG*ASE+AIE)*W2**2*2.*ZETA2*W2/AIXX
A(5,4)=(ALCG*ASE+AIE)*W2**2*W2**2/AIXX + W2**2*(-C2-AK3*ASE/AIXX)
A(5,6)=-C1
A(5,8)= -(FC*(ALCG*YPB(1)+YB(1)))/AIXX
A(5,10)= -(FC*(ALCG*YPB(2)+YB(2)))/AIXX
A(5,12)= -(FC*(ALCG*YPB(3)+YB(3)))/AIXX
A(5,14)= -(FC*(ALCG*YPB(4)+YB(4)))/AIXX
A(7,2)=(ASE*YB(1)-AIE*YPB(1))*W2**2*W1**2/GM(1)
A(7,3)=(ASE*YB(1)-AIE*YPB(1))*W2**2*(-2.*ZETA2*W2)/GM(1)
A(7,4)=(ASE*YB(1)-AIE*YPB(1))*W2**2*(-W2**2)/GM(1)+W2**2*RP*YB(1)
1/GM(1)
A(7,7)=-2.*ZETA(1)*WB(1)
A(7,8)=-WB(1)**2
A(9,2)=W2**2*(ASF*YB(2)-AIE*YPB(2))*W1**2/GM(2)
A(9,3)=W2**2*(ASE*YB(2)-AIE*YPB(2))*(-2.*ZETA2*W2)/GM(2)
A(9,4)=W2**2*(ASE*YB(2)-AIE*YPB(2))*(-W2**2)/GM(2) +
1 W2**2*RP*YB(2)/GM(2)

```

```

A(9,9)=-2.*ZETA(2)*WB(2)
A(9,10)=-WB(2)**2
A(11,2)=W2**2*(ASE*YB(3)-AIE*YPB(3))/GM(3)*W1**2
A(11,3)=W2**2*(ASE*YB(3)-AIE*YPB(3))/GM(3)*(-2.*ZETA2*W2)
A(11,4)=W2**2*(ASE*YB(3)-AIE*YPB(3))/GM(3)*(-W2**2) +
1 W2**2*RP*YB(3)/GM(3)
A(11,11)=-2.*ZETA(3)*WB(3)
A(11,12)=-WB(3)**2
A(13,2)=W2**2*(ASE*YB(4)-AIE*YPB(4))/GM(4)*W1**2
A(13,3)=W2**2*(ASE*YB(4)-AIE*YPB(4))/GM(4)*(-2.*ZETA2*W2)
A(13,4)=W2**2*(ASE*YB(4)-AIE*YPB(4))/GM(4)*(-W2**2) +
1 W2**2*RP*YB(4)/GM(4)
A(13,13)=-2.*ZETA(4)*WB(4)
A(13,14)=-WB(4)**2

C
C COMPUTE PHI(T), THE STATE TRANSITION MATRIX
C ITER=NUMBER OF TERMS USED IN TAYLOR SERIES EXPANSION TO CALCULATE PHI(T)
C
N=14
ITER=50
T=1./8.
DO 18 I=1,N
DO 18 J=1,N
PHIT(I,J)=0.0
PHITI(I,J)=0.0
18 W(I,J)=0.0
DO 19 I=1,N
19 W(I,I)=1.0
DO 3 I=1,N
DO 3 J=1,N
PHIT(I,J)=W(I,J)+PHIT(I,J)
W(I,J)=W(I,J)*T
3 PHITI(I,J)=W(I,J)+PHITI(I,J)
DO 4 K=2,ITER
CALL MULMAT(A,W,Z,N,N,N)
DO 4 I=1,N
DO 4 J=1,N
PHIT(I,J)=Z(I,J)+PHIT(I,J)
W(I,J)=Z(I,J)*T/FLOAT(k)
4 PHITI(I,J)=W(I,J)+PHITI(I,J)
WRITE(7,100)
WRITE(7) PHIT
100 FORMAT (12H PHIT MATRIX)
C
C BD(I) IS THE PRODUCT OF THE INTEGRATED PHI(T) MATRIX AND THE B MATRIX
C
DO 27 I=1,N
27 BD(I)=0.0
DO 28 I=1,N
28 BD(I)=PHITI(I,1)
DO 102 I=1,N
102 PRINT 103,I,BD(I)
103 FORMAT(10X,3HBD(,12,2H)=,E20.8)

```

```

WRITE (7,101)
WRITE (7) BD
101 FORMAT (10H BD MATRIX)
DO 29 I=1,N
29 X(I,1)=0.0
X(6,1)=10.0
DO 999 M=1,251
PHID=X(6,1)+YPD(1)*X(8,1)+YPD(2)*X(10,1)+YPD(3)*X(12,1)
1 +YPD(4)*X(14,1)
CONTRL=PHID*GC
CALL DIGCOM(CONTRL,BETAC)
888 PRINT 35,M,PHID,BETAC,CONTRL,XP(1,1),CX(1)
35 FORMAT (1H ,I5,1P9E13.4)
CALL MULMAT(PHIT,X,W,N,N,1)
777 DO 34 I=1,N
34 X(I,1)=W(I,1)+BD(I)*BFTAC
999 CONTINUE
GO TO 50
16 STOP
END
$IBFTC DIGCOM
SUBROUTINE DIGCOM(UI,YP)
COMMON/COM1/XP(3,2),BQ(1),C(2)
AO=0.0
A1=1.875
A2=-1.75
B1=-1.125
B2=0.375
FX=256.
UI=UI/3.0*FX
EX=1.0
CALL ROUND (UI,EX,FX)
UP=UI
BQ(1)=UP*3.0/FX
UI=UI*3.0/FX
IF(ABS(UP)-16.0) 2,3,3
2 C(2)=0.0
GO TO 4
3 C(2)=1.0
UP=UP/16.
IUP=UP
UP=IUP
FX=16.
4 IF(C(2)-C(1)) 5,6,7
5 XP(1,1)=16.*XP(1,1)
XP(2,1)=16.*XP(2,1)
AX=4.0
BX=63.75
CALL ROUND(XP(1,1),AX,BX)
CALL ROUND(XP(2,1),AX,BX)
GO TO 6
7 XP(1,1)=XP(1,1)/16.0
XP(2,1)=XP(2,1)/16.

```

```

AX=4.0
BX=63.75
CALL ROUND (XP(1,1),AX,BX)
CALL ROUND (XP(2,1),AX,BX)
6  XP(1,2)=-B1*XP(1,1)-B2*XP(2,1)+UP
   XP(2,2)=XP(1,1)
   AX=4.0
   BX=63.75
   CALL ROUND(XP(1,2),AX,BX)
   YP=(A1-AO*B1)*XP(1,1)+(A2-AO*B2)*XP(2,1)
1   +AO*UP
   CX=64.
   DX=255./64.
   CALL ROUND(YP,CX,DX)
   DO 1 I=1,2
1  XP(I,1)=XP(I,2)
   C(1)=C(2)
   YP=YP/FX*3.0
   RETURN
   END

```