

UNCLASSIFIED

AD 413321

DEFENSE DOCUMENTATION CENTER

FOR

SCIENTIFIC AND TECHNICAL INFORMATION

CAMERON STATION, ALEXANDRIA, VIRGINIA



UNCLASSIFIED

NOTICE: When government or other drawings, specifications or other data are used for any purpose other than in connection with a definitely related government procurement operation, the U. S. Government thereby incurs no responsibility, nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use or sell any patented invention that may in any way be related thereto.

AD No. 413321

ASD-TDR-63-158

DDC FILE COPY

DIGITAL DIFFERENTIAL ANALYZER

TECHNICAL DOCUMENTARY REPORT ASD-TDR-63-158

JUNE 1963

AF AVIONICS LABORATORY
ELECTRONIC TECHNOLOGY DIVISION
AERONAUTICAL SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO

Project No. 4421, Task No. 50920

Prepared under Contract AF 33(616)-6936, by
Tron Systems, Inc., Guidance and Control Systems Division,
Woodland Hills, California. Author: H. Baubrook

NO OTS

NOTICES

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

ASTIA release to OTS not authorized.

Qualified requesters may obtain copies of this report from the Armed Services Technical Information Agency, (ASTIA), Arlington Hall Station, Arlington 12, Virginia.

Copies of this report should not be returned to the Aeronautical Systems Division unless return is required by security considerations, contractual obligations, or notice on a specific document.

FOREWORD

This report was prepared by Litton Systems, Inc., Guidance and Control Systems Division, Woodland Hills, California, under Air Force contract AF 33 (616)-6936; task No. 50920; project No. D (613)-4421; titled "Development of An Airborne High Speed Digital Differential Analyzer".

The studies presented herein were begun in March, 1960, and the initial phase of the program was completed in May, 1961. The final phase of the program was begun in July, 1961, and was completed in November, 1962.

Mr. H. Banbrook was the principal investigator in the program and performed the bulk of the analytical studies. Mr. Banbrook also directed the technical efforts of supporting contributors to the program who, with their respective areas of interest, are listed below.

Equipment Program: L. R. Roth, Project Engineer

F. Rosenbloom, Logical Design

Logical Design of The DD³A: A. Whitehorn

Dr. G. Matthews

This report is a final report for the program, and concludes the effort under this contract. The contractor's report number is 2305-27.

The Air Force Project Engineer for this program was Captain Aubrey Calton, of the ASD Electronic Technology Laboratory.

ABSTRACT

The contract study resulted in new and powerful processing and mechanization techniques that have been integrated in the design of a full scale programmable multi-increment computer for full aerospace missions. Analyses of the fundamentally disparate processing requirements of essential aerospace subroutines demonstrated that radically new digital computer design techniques and more efficient computer mechanizations were required. Mechanizations were optimized for the basic types of computations, then integrated using modal design concepts to achieve a simplified system through new time-sharing techniques. Initial basic theoretical analyses formed the foundations for the integrative efforts. Numerical Stieltjes integration algorithms were derived and developed for input processing and internal computations. Fundamental digital Stieltjes algorithms attained new levels of accuracy in single and multi-increment computers. Invention of second difference computation and communication led to the first general ((quotient)) algorithm with multi-increment accuracy and a multi-transfer unit which, in cases, equals precision of conventional devices off twice the complexity. In meeting demanding aerospace requirements, serial-parallel arithmetic modal techniques were developed which enable a near continuous maximization of arithmetic capability for disparate routines off the full program, leading to decreased computer complexity for the required computation capability.

Best Available Copy

PUBLICATION REVIEW

This technical document has been reviewed and is approved.

FOR THE COMMANDER:

James R. Moore
JAMES R. MOORE
Acting Chief, Biometrics Branch
Electronic Technology Division

TABLE OF CONTENTS

Chapter	Page
Abstract	ii
I Introduction	I-1
III The Strap-Down Processor Constructed to Demonstrate Input Processing Principles and Special Purpose Mechanizations for Aerospace Applications	II-1
III Analytical Developments During Phase II in the General Theory of Real Time Computer	III-1
IV Design Concepts for Computer Systems with Pro- grammable Input Processing Capability	IV-1
V Evaluation of Auxiliary DDA Design and Computation Features	V-1
VI Development of Serial-Parallel DDA Mechanizations with High Duty Factors which are capable of Quotient Algorithm with Derived (Single Increment) Ternary and Later Developed Multi-Increment Computation	VI-1
VII New Concepts of Multi-Increment Computation and Development of a Multi-Increment General (Quotient) Algorithm Computer with Second Difference Outputs having Simplified Communication	VII-1
VIII Type of Programmable Modal Action of the Full Scale Incremental Computer implied by Computation Task and Mechanization Factors	VIII-1
IX DDA and QDD ² A Simulations on the IBM 704 Computer and Primary Results	IX-1

TABLE OF CONTENTS (Continued)

Chapter		Page
X	Quantitative Evaluation of Computation and Rate Handling Capability of DDA and QDDA Mechanizations	X-1
XI	Multi-Increment QDD'A Programs for Important Applications and Programming Efficiency Evaluation	XI-1
XII	Programmable Transfer Operations of the QDPU and QDPC Programming Code Studies	XII-1
XIII	Logical Design Investigations of Second Difference Incremental Computers with Conventional and General (Quotient) Algorithm	XIII-1
XIV	The Future Role of the Incremental Computer in Full Scale Aerospace Computer Systems and Proposed Study Efforts	XIV-1
XV	Brief Summary of Accomplishments of the HSDDA Study Effort	XV-1

LIST OF ILLUSTRATIONS

Figure		Page
2-1	Minimal Equipment Strap-Down Computer	III-25
2-2	Input Unit	III-30
2-3	Input Accumulator Unit	III-31
2-4	Information Flow in Input Accumulator Unit	III-33
2-5	Output of FAR 03	III-33
2-6	Multiplier Unit	III-34
2-7	Multipliher Out put	III-34
2-8	Multiplication Schedule	III-35
2-9	Extrapolator Unit	III-37
2-10	Output Unit	III-39
2-11	Information Flow in Output Unit	III-40
2-12	Drum Line Inputs	III-41
2-13	Output Organization	III-42
2-14	Bit Counter	III-44
2-15	Word Counter	III-45
2-16	Instruction Register	III-46
2-17	Output Cycle	III-51
2-18	Ready for Instruction	III-53
2-19	Fill Core	III-54
2-20	Serial Drum Connections for Drum Fill	III-58
2-21	Fill Drum	III-59
2-22	Set Address	III-60
2-23	Computer	III-71
2-24	Input Tape	III-72
2-25	Drum Fill Information	III-73
2-26	Input Tape	III-74

LIST OF ILLUSTRATIONS (Continued)

Figure	Page
2-27 Input Data Organization	II-65
2-28 Output Tape	II-66
2-29 Output Tape	II-67
2-30 Laboratory Model of the HSDDA Computer	II-68
2-31 Manual Control Unit for the HSDDA Computer	II-69
2-32 Information Processing for Evaluation of High Speed DDA Hardware	II-70
2-33 General Flow Chart	II-81
2-34 Central Differences for ΔU^*_{jk}	II-86
6-1 Register Array for Division in One Word Time	VI-14
6-2 QDPU Program Diagram for Updating x, y, and z with 100 Percent Efficiency	VI-19
6-3 QDDA Program Diagram for Toss Bombing	VI-21
9-1 Schematic of Simulated DDA Reciprocal Calculation	IX-12
9-2 Simulated Conventional DDA Calculation of Reciprocal Using Ambile's Method, Integration Algorithm Classically Mechanized	IX-14
9-3 Alternative Reciprocal Calculations	IX-17
9-4 Servoed Ambile's Method	IX-18
9-5 Iteration Number (m^*) Preceding Non-zero X_n	IX-22
II-1 QDPU Processing Schematic	XI-12
II-2 Missile Velocity Calculation With Scale Factor; Linear Drift and Bias Error Correction of Inputs and Output of the Digital Computer	XI-16
II-3 Complete Geographic Coordinate Calculations By the Proposed QDPU Multi-Increment, Without Double Integration Mode	XI-35

LIST OF ILLUSTRATIONS (Continued)

Figure		Page
11-4	Geographic Coordinates Calculation by the QDPU (Multi-Increment and Assuming a Double Integration Mode Not Incorporated in Final Design)	XI-27
11-5	Angular Rate Computation Program for the QDDA	XI-33
11-6	Gravity Computation Program for the QDDA	XI-36
11-7	QDPU Program for Thrust Cut-Off by Multi-Iteration Rate Computations of Input Processing and Internal Computation Routines	XI-45
11-8	Strap-Down Computation QDPU Program (Craft Orientation in Inertial Space)	XI-49
11-9	Strap-Down Computation QDPU Program Type for 3 of 5 QDPU	XI-50
11-10	Strap-Down Computation QDPU Program for 2 of 5 QDPU	XI-51
11-11	Schematic of the QDPU Program for Pitch Command During Re-Entry	XI-55
11-12	Doppler Damping Without GP Supervision (Multi-Increment QDPU Using Decision Mode)	XI-69
11-13	Doppler Damping Without GP Supervision (Multi-Increment QDPU Using Decision Mode)	XI-70
12-1	QDPU Register Labeling	XII -3
13-1	Arithmetic Mechanization of the DD²A Integrator	XIII-4
13-2	Integrator With the X Register Added	XIII-5
13-3	Integrator Word Structure	XIII-5

LIST OF ILLUSTRATIONS (Continued)

Figure		Page
13-4	Final Form of the Integrator Including an Associated Marking Channel	XIII-9
13-5	DD^2A Memory Structure	XIII-10
13-6	PDD^2A Output of the Second Differential Product of Xy	XIII-22
13-7	Elementary QDD^2A	XIII-23

CHAPTER I

INTRODUCTION

The study program at the conclusion of the second phase of a two-year effort resulted in development of markedly powerful processing and mechanization techniques which have been integrated in the design of a full scale programmable incremental computer system for full aerospace mission. Especially efficient mechanization for a computer has been derived which meets computation requirements for an aerospace mission from boost through coast to apogee, injection into orbit, retrofire, transfer orbit, re-entry, maneuvering and landing. The design level also provides for sophisticated auxiliary functions of military missions which may be expected within 5 years. It was demonstrated analytically and in simulations that the breadth of computation capacity required by the large set of computation routines, of individually fundamental disparate natures, is not met by existing real time computer systems of acceptable weight and size. To fulfill the goals of the contract effort, the following sub-investigations were carried out during Phase II (in considerable over-lapping time sequence): (1) applications surveys and evaluations were made of the computation requirements of the subroutines for these various applications; (2) a fundamental classification of computation types was carried out; (3) study of system implications with respect to overall computation capability and hardware requirements was completed; (4) improved digital computation techniques including new arithmetic modal design concepts and special basic transfer units were developed for hardware economy; (5) individually optimum mechanizations for each computation type were integrated i. e., arithmetic modal design techniques were developed for time shared minimized mechanization to form a single versatile mechanization for the required new level of computation capability in a computer of moderate hardware complexity.

Phase I had consisted of a broad study ranging from the analysis of the basic nature of the most demanding airborne guidance and control computation problems, and the general computation approaches to their solution, to the development of specific high performance incremental computation algorithms. To show the computer design techniques and evaluate a theory of numerical Stieltjes integration that was developed, a special purpose computer for strap-down computations was designed during Phase I and later constructed during Phase II.

The general approach to full scale computer system design for aerospace and airborne applications, of which the strap-down processor exemplified but a major part, was seen to offer a potentially great step in computation capability for a full scale system of given level of mechanization and complexity, provided major developments in digital computer design techniques exploited them in full measure. The well defined goal of Phase II made possible an intense concentration on the challenging design problems involved and assisted in development of the design of a full scale incremental computer system which provides a remarkably higher level of computation capability for given hardware costs than existing computers constructed from the same state of the art hardware. Design techniques were developed for extracting consistently in real time greater real computation value. The basic classes of computations which comprise airborne and aerospace computation programs, introduced as input processing and internal computation in the analyses of Phase I, are provided for in a highly integrated programmable incremental computer evolved during Phase II. Further detailed analyses during Phase II revealed that certain additional requirements (to be described), are not met in existing incremental computers, but are needed in a full aerospace mission computer. However, these and previously recognized requirements are met in highly efficient design which utilizes the advanced algorithm and digital processing techniques developed during Phase II of the program.

Analytical efforts during Phase II in the general theory of numerical incremental computation, served to complete the establishment of the general relationships of the processing concepts, with development of simplified algorithms for internal computation in terms of what is referred to as "virtual" variables.

Computation requirements and capability analyses for a full aerospace mission demonstrated that for internal computations the computer with the time shared arithmetic module, requires (1) several bit increment computation, (2) precision general (quotient) integration algorithm and (3) a degree of parallel processing capability. These sophistications are shown realizable without significant additional cost, assuming a programmable input processing capability, because of the concept of a single arithmetic module, that is simpler than a whole-word fast multiplier and that uses time sharing and modal switching in the execution of all the system functions each of which is at an individually required level of precision. In the internal computation design studies a major deficiency of the conventional DDA, but a partially developed attribute of at least one of the more recent incremental computers, was fully developed upon discovery of digital processing techniques capable for the first time of division with multi-increment accuracy in an incremental computer (with no direct division capability). Because of high variable rate data and precision requirements, division algorithm is amenable to accurate doppler damping in conventional navigation, in coordinate transformation computations, such as toss bombing and fire control, and, most important, in the generally demanding aerospace computations.

The multi-increment computer is almost universally considered inherently to have a more complex communication structure, but new techniques based on the theory of information for band limited variables led to a computer structure in which communication costs are less than in a conventional

single increment DDA with the same integrator count. Multi-transfer unit costs are reducible to an entirely acceptable proportion of computer system costs. For applications demanding both input processing, as well as internal computation capability, a time shared arithmetic module capable of parallel (high rate) several bit increment and serial (intermediate rate) many bit increment computation was evolved. Another remarkable multi-transfer unit was developed for internal computation by applying the new design techniques. The new multiplier is capable of M bit transfer, with mechanization comparable to a three bit transfer unit, where, for example, M may be as large as 10 bits depending on the scaling implied by the analytic or empirically determined character of the computation variable.

The development during phase I of this contract study of a design technique for general (quotient) algorithm computation with multi-increment accuracy had not been met for challenging technical reasons. The solution during Phase II of this problem for hand-limited variables was accomplished together with invention of second difference computation and communication in what is believed to be a significant contribution to the field of digital computation. In addition to overcoming this design problem the techniques developed directly imply remarkable simplifications in general multi-increment computer mechanization.

Integration of the many new processing techniques into a single system was accomplished in a natural and highly efficient manner. The result was a design structure of an incremental computer of modest weight and volume, assuming use of state of the art hardware, and modest clock rate, which can execute the computations on continuous variables (and with proposed design developments, piecewise continuous variables) involved in the many tasks including thrust cutoff, strap-down computation, guidance and control in all phases of the aerospace mission including re-entry with sophisticated energy management and all at the new levels of accuracy required.

CHAPTER II

THE STRAP-DOWN PROCESSOR CONSTRUCTED TO DEMONSTRATE INPUT PROCESSING PRINCIPLES AND SPECIAL PURPOSE MECHANIZATIONS FOR AEROSPACE APPLICATIONS

2.0 INTRODUCTION - Input processing principles developed during Phase I consisted of principles for Stieltjes numerical integration, multi-increment, and multi-iteration rate computation of integral increments. The strap-down processor was constructed during Phase III to demonstrate the high level of accuracy attainable by application of those principles to a computation problem with high error sensitivity, such as that of strap-down computations. It was proposed that on the basis of these and expected further developments, in processing and mechanization principles, that a full scale design be developed for a computer capable of executing all aerospace computation tasks of a full mission, and mechanizable within a package of modest size and weight. This chapter presents contract study results which, rather than a direct general stage of the primary effort, are the results of special analyses, hardware design and evaluation associated with the strap-down processor and special purpose strap-down computer design. The strap-down processor is a special purpose computer which, quite apart from input processing principles, incorporates special purpose computer design principles. While the primary contract study effort was devoted to the development of a full scale programmable computer system, it was nevertheless considered profitable to evaluate the implications of the strap-down processor as a design basis for special purpose computers which execute strap-down computations. Certain applications, such as pre-mid course missile guidance, are regarded by many system analysts as calling for a strap-down computer, a special purpose computer to be placed in an intermediate missile launch stage package.

For such applications the strap-down computer, for a system using state of the art sensors and transducers, is tailored to meet a given precision requirement. This presents the relatively straightforward design task of modifying specific parts of the strap-down processor to adjust multi-increment bit length, algorithm sophistication, and input accumulation for the transducer type to meet the generally less demanding levels of those applications.

2.1 STRAP-DOWN PROCESSOR DESIGN MODIFICATIONS DURING PHASE II AND ANALYSIS OF RESULTANT COMPUTATION CHARACTERISTICS.

- A. Introduction -** Certain design modifications to the preliminary design of Phase I were made during Phase II to obtain a special purpose computer capable of not only high precision, but one whose design could be readily tailored to possible special purpose applications.
- B. Strap-Down Processor Design Modifications -** The input processor logical design problem, involved in the sequencing of component calculations of the angular and inertial velocity requirements, was reviewed to establish a process which achieves total updating within a single slow iteration interval. The modified logical design, adopted and described in the section on logical design, achieves a sequencing with simpler input processor mechanization and checkout, and has an output variable set more naturally assimilated by an internal computer or output device. A second major modification is the shortened bit length of the multiplier unit to 19 bits gained by introduction of a roundoff operation on input accumulator outputs to the multiplier. Simple roundoff based on the value of the 20th bit is shown in a later section to introduce bias error significant with respect to processor output accuracy, the effects of which can be removed in the following alternative modifications:

1. Biasing of analog input reference voltage level.
2. Logical design of a somewhat sophisticated roundoff process.

The latter was chosen in the final logical design.

2.2 SCALING PROPERTIES OF THE STRAP-DOWN PROCESSOR AND MINOR HARDWARE MODIFICATIONS FOR UNUSUAL APPLICATIONS -

The basic design features of the High Speed Digital Differential Analyzer (HSDDA) for strap-down computations are such that, essentially, any real strap-down computation application can be handled provided that in certain cases, minor hardware modification is made. The rate handling capability is the result of the whole word input feature. The minor modification which in certain applications may be necessitated is the result of non-programmable internal scaling, specifically in the phasing of updating additions determining magnitude of increments of direction cosines. The modification which easily attains any conceivably desired rate handling capability is a delay of extrapolator output by d bit times to attain a 2^d increase in rate handling capability. Anticipated future applications do not require more than a 2 bit time delay.

Consider now the details of internal scaling of the strap-down processor. The 20 bit input words which are angular increments and velocity increments are accumulated 9 times in pre-processing. The immediately subsequent input extrapolation is mechanized with an effective increase in amplitude of 8 (the result of a 3 bit time delay). The resulting quantities are rounded off and effectively decreased by a factor of 2^{-7} on entering the multiplier as 20 bit numbers. The scale of these multiplier quantities is seen to be

$$9 \times 8 \times 2^{-7} = 9/16 \quad (\text{II-1})$$

relative to the input words. Assuming the U_{jk} quantities have scale S_U the output of the multiplier has scale $9/16 S_U$. On passing through the extrapolator the scale is increased by a factor of 12. The outputs of the extrapolator are presently added to the U_{jk} line at the least significant end of the 32 bit word. Since the 32 bit word for U_{jk} quantities is regarded in multiplication as unity for full register the latter updating mechanism amounts to a scale factor reduction of $2^{-31+19} = 2^{-12}$. The scaled U_{jk} quantities make this part of the updating effective by a scale factor change of $2^{-12}/S_U$. The net relative magnitude of ΔU_{jk} to U for full input angular rate is given by:

$$\frac{9}{16} S_U \cdot \frac{12 \times 2^{-12}}{S_U} = \frac{27}{32} \cdot 2^{-9} \quad (\text{II-2})$$

which is the largest fractional change in U which can be carried in one iteration (without the minor modification previously discussed). The maximum angular rate which can be handled without the proposed change is

$$\omega_{\max} = \frac{27}{16(1024)} (256 \text{ it/sec}) = 43 \frac{\text{rad}}{\text{sec}} \quad (\text{II-3})$$

for the 266 it/sec iteration rate of the HSDDA. The simplest method of increasing ω_{\max} is to displace the write head of the U channel the proper number of bits to increase ω_{\max} a factor of 2 for each bit moved over.

2.3 ROUND OFF ERROR GROWTH IN THE STRAP-DOWN PROCESSOR.

A. Introduction - The final logical design of the strap-down processor implies a computer subject to roundoff errors at only three points of the input processing. These are:

1. Input conversion, resulting from sampling inputs to finite word length.
 2. Roundoff of Input Accumulator Outputs, enabling use of a fast multiplier of moderate word length.
 3. Roundoff in the multiplier, which is negligible because the Litton fast multiplier effects roundoff at double word length.
- B. Roundoff Error Resulting From Input Sampling - Input sampling errors are assumed to be reduced to purely random errors by elimination of converter bias errors. The 20 bit word input of 19 bits magnitude plus a sign bit are then sampled with a resultant maximum error value of $\frac{1}{2} \times 2^{-19}$ of full scale. A flat probability distribution implies that the rms error of a single converted value is $\frac{1}{\sqrt{3}} \times 2^{-20}$ of full scale. The maximum angular rate for full scale of inputs is adjustable in the strap-down processor, being in the shipped computer 0.43 rad/sec for which the rms angular rate error of a single reading is

$$c_s = 0.43 \times \frac{1}{\sqrt{3}} \times 2^{-20} \approx \frac{1}{4} \times 10^{-6} \text{ rad/sec} \quad (\text{II-4})$$

The cumulative random error in angle produced in an inertial coordinate may be evaluated on the basis of summation at the input rate of $1/\tau = 2.00$ iter/sec which is 9 times the output rate. The angular error sum after a period of operation τ is

$$\theta = \sum_{n=1}^{\tau/\tau} \epsilon_{\theta,n} \tau \quad (\text{II-5})$$

which has variance

$$\sigma_{\theta}^2 = \tau^2 \sum_{n=1}^{t/\tau} c_{e_{w_n}}^2 \quad (\text{II-6})$$

Hence

$$\sigma_{\theta}^2 = \tau^2 \frac{t}{\tau} c_{e_w}^2 \quad (\text{II-7})$$

$$\sigma_{\theta}^2 = \sqrt{t \tau} c_{e_w} \quad (\text{II-8})$$

For $t = 6 \text{ hr}$, $1/\tau = 2400$, $c_{e_w} = 1/4 \times 10^{-6}$ the rms error resulting from input sampling is approximately $3/4 \times 10^{-6} \text{ rad} \approx 1/6 \text{ arc sec}$, hence the random input sampling error is negligible.

- C. Error Growth Resulting From Roundoff of Multiplier Inputs - The hardware requirements of a whole word fast multiplier are essentially proportional to the word length of the multiplicand. A substantial saving in multiplier complexity was effected by introducing a roundoff operation on outputs of the input accumulator unit (operation of which is delineated in the section on general logical description of the strap-down processor). The basic inputs to the strap-down processor are 20 bit words of 19 bits magnitude plus a sign bit. A constant input is essentially multiplied by 9 in pre-processing summations and then multiplied by 8 in the input extrapolation section hence the input accumulator outputs before roundoff can be 27 bit words of 26 bit magnitude plus a sign bit. The roundoff operation, producing a number of 20 bits if sufficiently accurate, may introduce an acceptable error

of the same low level as in input sampling. The most commonly mechanized roundoff operation, however, can be shown to be inadequate because it introduces a bias of 2^{-N+1} where N is the number of bits rounded off. The simple roundoff operation consists of adding a 1 in the most significant bit rounded off, to the truncated number in the least significant bit position. The average error introduced in this operation is evaluated by assuming that all possible numerical values of the truncated number occur with the same frequency during extended computer operation, and therefore average the individual errors which result for each possible number. The truncated number, regarded as in a unit register, ranges from 0 to $(1-2^{-N})$ and includes 1/2. The value 0 introduces no error when it occurs. The remaining successive values, on each side of 1/2 of each absolute magnitude difference from 1/2, produce equal and opposite errors. The resultant error using the simple roundoff operation is purely that introduced when 1/2 occurs, for which the error is 1/2. Since the number 1/2 occurs with frequency 2^{-N} the long term effect of the simple roundoff method is a bias error of 2^{-N} times the least significant result of the rounded number. In the strap-down processor the simple roundoff would create an effective bias of 2^{-7} of the least significant bit of the multiplicand. For full scale direction cosines the least significant bit passes directly through the multiplier, thence through the extrapolator unit where it is multiplied by 12 and then added at the least significant end of a 32 bit word of the direction cosine line. Regarding the error produced in the direction cosine line as an angular error in radians (since for θ small $U = \cos(\theta + 90^\circ) \approx \theta$ the angular error produced per iteration (on the average) is:

$$2^{-31} \times 12 \times 2^{-7} \approx 0.5 \times 10^{-10} \text{ rad} \quad (\text{II-9})$$

After 6 hours at 266 iter/sec the bias errors can add up to a total angular error of

$$6(3600)(260) 0.5 \times 10^{-10} \approx 3 \times 10^{-4} \text{ rad} \quad (\text{II-10})$$

$\approx 1 \text{ arc min.}$

resulting from the use of the simple roundoff operation. The bias error can be removed using the elaborated roundoff operation based on the fact that error is produced when the roundoff section of bits, regarded as in a unit register, has value 1/2 in which case the addition of a bit to the truncated number in the simple roundoff is now inhibited. With effective bias error removed the only error effect remaining is the random error effect analyzed in the previous section.

2.4 LEAD-LAG EFFECTS IN INPUT SAMPLING AND ASSOCIATED ANALOGUE FILTERS - Strap-down computations are extremely sensitive to lead-lag effects in input variables or processing action. The analysis of Phase I presented in Chapter 7, Section 4, of the first report,* showed that introduction of an analogue filter of very short time lag between sensor and digital computer was necessary in order to effect the second order Stieltjes integration algorithm in a strap-down processor, with simplified mechanization. Assuming the strap-down processor mechanization executes precisely the processings derived in that analysis the time constant of the analogue filter was shown to

* First Phase Technical Documentary Report on Development of an Airborne HSDDA, H. W. Banbrook, 7 July 1961. Contract AF33(616)-6936.

necessarily be $\frac{17}{72}\tau$ where τ is the output iteration interval. In the final mechanization the processing was modified slightly to imply analogue to digital conversions made by only two converters, rather than six converters (one for each input variable) for a hardware saving in any operational system. The effect of serial rather than parallel samplings of inputs is to introduce leads and lags in certain angular rate and acceleration inputs. The leads and lags are ± 1 word times of $\pm 1/27\tau$ which while small would have a serious effect on computer accuracy if not corrected in an operational system. Such correction made for each input in the associated analogue filter, by choosing the time constant to be $\frac{17}{72}\tau \pm \frac{1}{72}\tau$ instead of $\frac{17}{72}\tau$, is evaluated since such correction involves no actual cost in mechanization. A small error effect in the second order algorithm results from making the parameter change for exact first order algorithm. The level of error in second order algorithm for the case of 1 word time phasing error is seen by the following analysis for the general case of W word times. Expressed in the delay operator form of previous algorithm analysis the necessary compensation of a W word time delay in the 27 word per iteration processor is

$$(1-\zeta)^{-\lambda} \approx (1 + \lambda\zeta + \frac{\lambda(\lambda+1)}{2}\zeta^2) \quad (II-11)$$

where $\lambda = \frac{W}{27}$, and ζ is the 1 iteration delay operator. The required compensation for an analogue filter was shown to be

$$1 - u^* \zeta - u^* (\frac{1}{2} - u^*) \zeta^2 \quad (II-12)$$

where $u^* = 1/K^*\tau$, and K^* the time constant of the modified filter intended to compensate the lag of the sampling.

The net algorithm effect is given by the product of the required compensations, which should approximate the required compensation for the filter with time constant $K = \frac{72}{17}$, determined for the $W = 0$ case. Thus the net uncompensated algorithm effect is given by:

$$A = \frac{\left[1 + \lambda\zeta + \frac{\lambda(\lambda+1)}{2}\zeta^2\right] \left[1 - u^*\zeta - u^*(\frac{1}{2} - u^*)\zeta^2\right]}{\left[1 - u\zeta - u(\frac{1}{2} - u)\zeta^2\right]} \quad (\text{II-13})$$

The lag cancellation yields first order agreement as implied by equating first order terms in numerator and denominator,

$$1 - u^* = 0 \quad (\text{II-14})$$

which on substituting for λ , u , u^* implies the choice

$$\frac{1}{K^*} = \frac{1}{K} + \frac{W}{27} \quad (\text{II-15})$$

The net uncompensated algorithm effect for $u^* = \lambda + u$ is

$$A = \frac{1 - u\zeta + u(\frac{1}{2} - u) + \lambda(\frac{1}{2} + u)\zeta^2}{1 - u\zeta - u(\frac{1}{2} - u)\zeta^2} \approx 1 + \lambda(\frac{1}{2} + u)\zeta^2 \quad (\text{II-16})$$

to second order

Substituting for λ , u ,

$$A \approx 1 + \frac{W}{27} \left(\frac{W}{54} + \frac{17}{72} \right) \zeta^2 \quad (\text{II-17})$$

which for 1 word time lag is

$$A \approx 1 + 0.0095 \zeta^2 \quad (\text{II-18})$$

The compensation of second order term to 1 percent corresponds to precision improvement due to second order algorithm relative to first order algorithm of 50 to 1. Note that if an additional filter were used in series with the required filter, for which the analysis above may be used, with $\omega = 0$, then

$$A \approx 1 + \left(\frac{W}{27}\right)^2 \zeta^2 \quad (\text{II-19})$$

which for $\omega = 1$

$$A \approx 1 + 0.004 \zeta^2 \quad (\text{II-20})$$

indicating negligible second order algorithm error, using a two filter per input, mechanization.

2.5 TRANSMISSION WITHIN THE STRAP-DOWN PROCESSOR OF SECOND ORDER ALGORITHM TERMS FOR HIGHER ORDER INTEGRATION ACCURACY -
 Roundoff procedures and word length determine the level of roundoff error. If the roundoff error exceeds the level of algorithm error produced by neglecting second order terms in the integration algorithm then, depending on the application, either the sophistication of higher order integration is not justified, or higher precision computation is required. Employing the most precise roundoff techniques it is possible to compute effectively, including higher order effects, even if they are smaller than the resolution. This is seen in the case of first order algorithm computation in a single increment DDA in which the difference between first order algorithms in accuracy, in a sinusoid for example, is very great, yet the size of Δx in the algorithm may be much smaller than the single bit. In general however the noise in transmission of small terms makes the value of algorithm terms a significant number of bit positions smaller and the resolution essentially nil. Consider, for the case of the strap-down processor, the determination of the level of magnitude of second order algorithm terms compared to resolutions. The second order

Terms in the strap-down inertial orientation computations involve changes in angular rate between iterations. A feature of auto-piloted flight, especially in the atmosphere, is that craft axis angular rates change at usually several times the angular rate of the craft axes. This is explained by the fact that control of the craft about a fixed chosen orientation without complete rotations requires that the phase of angular rate changes must change before the angle of orientation has changed as much as say $< 1/3$ radian. In the Dutch roll case,

$$\omega = \omega_0 \cos \theta_0 t \quad (\text{II-21})$$

where

$$\theta_0^{\circ} / \omega_0 \approx 3 \text{ to } 6$$

then

$$\Delta\omega \approx -\theta_0^{\circ} \cdot \omega_0 \sin \theta_0^{\circ} t \quad (\text{II-22})$$

from which it is deduced that $(\Delta\omega)_{\max} / \omega_{\max} = \theta_0^{\circ} \tau$. Assume $\theta_0^{\circ} \approx 4 \omega_0$ in typical flight involving craft motions similar to Dutch roll. Then the rms magnitude of second difference terms in direction cosines is related to that of first order terms in a ratio of about $4 \omega_0$, where ω_0 is regarded as maximum input angular rate and τ is processor output iteration interval. In the Stickjes integration process of the strap-down processor the independent variable is angular displacement, scaled as a fraction of full input register, and has a maximum for 0.4 rad/sec of

$$\theta_0 = 4 \omega_0 \tau = \frac{4(0.4)}{260} = 0.0061 < 2^{-7} \quad (\text{II-23})$$

At 0.1% of maximum angular rate the change of angular rate per iteration is less than the resolution of the input register having 19 bit and sign, but is expected to have accurate statistical transmission, assuming precise sampling

techniques. The angular displacements (modified for algorithm) leave the input accumulator, and enter the multiplier where they are multiplied by 31 bit (plus sign bit) direction cosines, the products being rounded off to 19 bits. Entering the extrapolator, second differences contribute to the output according to a magnitude level determined by the second order terms of angular displacements, and direction cosines (and products of first order terms). For full scale of the direction cosines of 1° their second differences have maximum magnitude

$$(w_{0_{\max}} \tau) (\dot{f}_{0_{\max}} \tau) \approx 4 (r_{0_{\max}} \tau)^2 = 10^{-5} \quad (\text{III-24})$$

for $r_{0_{\max}} = 0.4 \text{ rad/sec}$, $\tau = 1/266 \text{ sec}$. Entering with weight $5/12$ in the extrapolator action their magnitude compared to the least significant bit of the 19 bit output from which they are computed is generally less than

$$2^{19} \times \frac{5}{12} \times 10^{-5} > 2 \quad (\text{III-25})$$

that is 4 times the resolution. Since the mechanism of the extrapolator unit introduces no error whatever on a perfect 19 bit input, the only source of error in second order term transmission would be in the output of the multiplier (and earlier stages of computation). Since the multiplier has very precise roundoff the second order terms when sub-significant may be regarded as fully transmitted but contaminated with roundoff noise in a pulse stream representation (in pulses of magnitude of the least significant bit) in which the frequency represents the magnitude. In Dutch roll (worst case) the overall effect of second difference terms on inertial reference is accumulative. The cumulative effect (reduced by a factor of two for phase effects) resulting from second order terms in 1 hour can be estimated by taking into

account the scale with which the terms are added to the direction cosine line (a more detailed analysis of Dutch Roll is presented in the final report of Phase I), the scale being $1^{\circ} \times 2^{-31}$, for which the maximum error is

$$\begin{aligned} & 1/2 \times 1^{\circ} \times 2^{-31} \times (2 \text{ bits}) \times 266 > 360 \text{ radians} \\ & = 5 \times 15^3 \text{ radians} = 17 \text{ arc min} \end{aligned} \quad (\text{II-2b})$$

Implementation of carefully chosen roundoff procedures is called for in computing with second order accuracy since this significant error effect can result from terms contributing magnitudes in 19 bit numbers at the least significant bit positions. Viewed purely from the standpoint of roundoff error the analysis of IIB4 showed that the required care has been taken in the strap-down processor design.

2.6 SPECIAL PURPOSE MECHANIZATIONS FOR AEROSPACE APPLICATIONS REQUIRING STRAP-DOWN COMPUTATIONS.

A. Introduction. - The marked performance and mechanization characteristics of state of the art sensors, and certain low cost transducers present constraints on digital computer design for aerospace applications in which computer weight and volume are significant. In the case of strap-down inertial reference computations the relatively low accuracy level of the state-of-the-art rate gyros (discussed in the next section) imposes an unusually marked design constraint. The direct impact of this constraint is the implication of a low cost transducer, the pulse stream analogue to digital converter, which has accuracy limitations (discussed in a later section) comparable to the rate gyros. The implications of the sensor and transducer accuracy constraints are

the limitation to a rather narrow subset of applications of strap-down inertial reference computations requiring modest accuracy. This fact implies an allocation of digital computation capability of reduced degree. Note that the implication regarding computer sophistication depends on whether the strap-down computations are the only task, or merely one of many tasks of the computer. For applications in which system analysis implies the need for a special purpose computer for simple strap-down inertial reference computations, to be executed in an individual unit in a missile system (physically separated from the main computer system), the requirement for a special purpose computer of modest size and probably modest mechanization complexity is seen to be clear cut. For applications in which the strap-down inertial computations or similar computations comprise only a part of the computation task, the mechanization of the computer may necessarily be of at least moderate sophistication because of the overall computation capacity required. In this section pertinent special purpose computer mechanization designs are developed in contrast to the primary design task consisting of the design of a full scale computer system for a full aerospace mission. The strap-down processor was constructed on the basis of the analytical developments of Phase I and provides a design basis for the special purpose computer with several applications. It also served as the basis for the subsequently developed programmable input processing portion of the primary design effort which was directed toward the development of a full scale computer for the full aerospace mission.

- B. State of the Art Sensor Limitations - State of the art rate gyros have accuracy limitations on the order of 0.01 percent of maximum

angular rate. Until markedly higher accuracy angular rate sensors are developed, the major application of strap-down inertial reference systems is limited to short period operation where very high accuracy is not important. Certainly, long period airborne navigation does not presently fall into this class of application. The strap-down processor developed in this contract study is capable of executing computations for any of the broad set of applications presently conceived in anticipation of the development of adequate sensors. The direct purpose of constructing the computer was to establish computation capability using the general design techniques developed during Phase I.

The basic limitation of the rate gyro, as an angular rate sensor, stems from the fact that an analogue voltage is generated to produce counter balancing torque on the rate gyro when the case is rotating relative to inertial space in order to cause the gyro to follow the case. The source of error is that common to all analogue electrical systems. The indicated path to major improvement in angular rate sensors appears to be in the direction of obviating the requirement of generating the torque signal by analogue voltages which in turn must be assumed proportional to the torque. It would appear that sensor development must necessarily rest on avoidance of this error source and must perhaps utilize some of the dramatic technical developments in the field of precision angle generation and measurement. An angular rate device using optical measurements also can obviate the analogue to digital transducer problem. Consider, for example, the microgon (the high accuracy angle encoding system which is a product of the Norden Division of United Aircraft) which is capable

of reading 10^6 counts per turn of an input shaft and of accurately following shaft angular rates corresponding to 175,000 counts per second. An angular rate of 1 radian per second can be read by the Microgon to 0.05×10^{-4} which is at least 20 times more accurate than inertial angular rate defined by a rate gyro. The Microgon output being a whole word, it is ideal in this respect for the strap-down processor input. The unsolved problem of making the input shaft of the Microgon maintain nil rotation with respect to inertial space in more than one degree of freedom is, however, an obstacle in the way of a definite solution.

C. Transducer Performance Limitations and Mechanization Costs.

1. Performance of the Pulse Stream Analogue to Digital Converter - The pulse stream type of transducer has relatively simple mechanization compared to that of the whole-word sampler (of say 14 to 20 bits). The information transmission capability of the former can be far less unless the pulse rate greatly exceeds the sampling rate. The quantitative computation error associated with the pulse stream transducer depends on the required information transmission of the application as well as the capability of the device. Thus, the band limited character of inputs implies the possibility of better performance than would otherwise be expected. The use of the pulse stream transducer for a specific application is tempered by the effective accuracy in relation to state of the art sensor accuracy. The lead-lag effects of a bias-corrected pulse stream transducer, which result from the ambiguity of sensor outputs in pulse stream representation, are possibly serious for strap-down computations

because of the sensitivity of the latter to such effects. These effects are exaggerated at very low rates, indeed, for motion within an angular range of the resolution they are capable of periods of bias on the order of the resolution.

2. A Preliminary Theory of Pulse Stream Transducer Error Effects Applicable to Strap-Down Computations - Extensive DDA sinusoid simulations during Phases I and II of this contract study led to the formulation of a theory for a particular roundoff effect which is the major type of round-off error for sinusoid computations. The overflow inhibitor design technique led to much less frequency infidelity (relative theoretical frequency) of sinusoid calculations. The error effect takes place when the output rate of a rounded variable is near null. A theoretically derived formula for the effect which is in quantitative agreement with micro and macro error magnitudes of sinusoid computations is applied in Chapter X in estimating roundoff error in general DDA calculations. The pulse stream transducer is a roundoff device which may induce error effects (among others) of the same nature as that evaluated. Thus, the design principle of the overflow inhibitor could be applied to substantially improve the performance of a system employing the pulse stream transducer, especially for the strap-down application.
3. Digital Computer Mechanization for Analogue to Digital Converter - A whole word sampler is assumed in the constructed strap-down processor. Assuming the use of the pulse stream transducer a significant implication in the digital computer design results because of the pulse rate

limit of the state of the art devices. Thus, for a strap-down processor with an iteration rate of 400 it./sec and a pulse stream transducer with a maximum 2×10^4 pulses/sec, the maximum number of pulses per iteration is 50, the sum of which may be represented by a 6 bit binary number. As a result of the character of the strap-down computation the digital multiplier unit appropriate for this application, without intermediate quantization (which lowers accuracy) is a 6 bit multiplier. The relatively large granularity imposed by the pulse stream transducer obviates input pre-processing (appropriate for the whole-word transducer). The pulse stream transducer can be mated with the strap-down processor by a summation register reset at each iteration to zero, the contents being processed, thereafter, in the same manner as sums of nine samples formed at high rate in the preprocessing of whole-word inputs in the bread-board strap-down processor. A hybrid transducer with pulse stream and short word inputs could be mated to the bread-board strap-down processor to retain the major mechanization simplification of the pulse stream transducer which is an overall reduction of converted word length. A hybrid mechanization which utilizes the pulse stream in parallel with a several bit analogue to digital conversion of the sub pulse analogue signal which triggers the pulse output is possible. The merit of such a transducer, of intermediate complexity, is the reduction of lead-lag sources in digital computation resulting with increase of effective word length.

2.7 SPECIAL PURPOSE STRAP-DOWN COMPUTER DESIGN TECHNIQUES

IMPLIED BY STATE OF THE ART SENSORS AND TRANSDUCERS - Previous discussions concluded that a special purpose strap-down computer for aerospace application need not achieve a computer accuracy comparable to that required for long term inertial navigation, specifically because of state of the art sensor accuracy limitations which limit applications to those with corresponding accuracy requirements. The relatively high error sensitivity of the strap-down computations (assuming no sensor error) however, implies a computer, even for present special purpose computer applications, of distinctly greater computation capability than state of the art incremental computers. The design of the constructed strap-down computer may be tailored (for these special purpose applications) to supply the required computation capability in a mechanization of minimum complexity. The straightforward tailoring procedure consists of:

1. Modifying the input accumulator to absorb pulse stream transducer information.
2. Reducing the multi-increment computation bit length, and obtaining further hardware economy by mechanizing the simplified multiplier, for which a design technique was developed during Phase I.
3. Reducing the complexity of the integration algorithm.
4. Reducing the word length to correspond to the resolution of the input pulse information.

Specific problems associated with these procedures are:

1. Input Accumulation With Pulse Stream Transducer Inputs - The pulse stream transducer supplies pulses of angular change (preferably at a maximum rate which is high for precision) to a computer with lower iteration rate (for simpler mechanization which avoids a high degree parallel processing). The appropriate input accumulator may have an identical processing form to that for whole-word sampling, however, it has (higher) pre-processing rate T_F (an integral multiple of output rate) which just exceeds the maximum input pulse rate, thus obviating appreciable buffering of inputs.

2. Multi-Increment Bit Length - Selection of inputs as independent variables of integration implies that the multiplier must have the capability of executing multi-transfer of bit length $\log_2 \tau/\tau_F$. The arithmetic unit may have fast multiplier mechanization for this bit length requiring one word time per multiplication or, if multi-transfer bit length ($\log_2 \tau/\tau_F$) > 5, it may have a multi-pass multiplier which may be more economically mechanized. The latter has fast multiplication capability which is only a fraction of the required multi-transfer bit length for one word time multiplication, and consequently implies a reduced processor iteration rate relative to that obtainable by the more costly multiplier. The case of pulse stream transducer with the maximum rate of about 10^4 pulses/sec for maximum angular rate of 1 rad/sec implies for inertial reference an inertial velocity computation with the same bit rate as the constructed input processor i.e. a multi-transfer bit length of 5 bits, a word length of 16 bits and an iteration rate of 500 it/sec using a fast multiplier. A two-pass three-bit multi-transfer multiplier computes at 250 it/sec with the same resolution.
3. Simplified Integration Algorithm - The inherent errors of the state of the art sensors and pulse stream transducers make computation with precise first order algorithm the natural choice in further simplifying mechanization. A less than first order algorithm, such as that employing old y instead of new y algorithm when the latter is appropriate, would seriously degrade overall accuracy. In a special purpose computer, based on the input processor, certain delay lines and adders are left out in realizing the simplified algorithm with some hardware saving.

2.8 STRAP-DOWN PROCESSOR MECHANIZATIONS TAILORED TO SPECIAL APPLICATIONS - As a result of differing strap-down processor accuracy and unit weight/volume requirements for different applications, three systems have been evolved each of which is appropriate for the particular class of

applications for which it was designed. The three computers are briefly described as below, and the minimal version is shown in Figure 2-1.

A. Minimal Equipment Strap-Down Computer.

1. Performance: Reference error $\leq 1^\circ$, velocity error $\leq 2 \text{ ft/sec}$ (in 15 minutes) for launch phases of missile guidance.
2. Analogue to Digital Converter: Angular and velocity increment pulse stream type with 0.002° granularity.
3. Integration Algorithm: One level higher accuracy than conventional DDA algorithm.
4. Multi-Transfer: 15 bit multi-transfer operation by 5 bit (multi-pass) multiplier.
5. Memory: Delay line (or drum).
6. Iteration Rate: 500 it/sec with delay line (100 it/sec with drum) memory.
7. Volume: 1/4 cubic foot with delay line (1 cubic foot with drum) memory.

B. Full-Scale Aerospace Strap-Down Computer.

1. Performance: Reference error $\leq .05^\circ$, velocity error $\leq 2 \text{ ft/sec}$ for launch and re-entry phases (in which a pre re-entry spot reference correction is made by stellar or other means).
2. Analogue to Digital Converter: Whole-word sampler of angular rates; granularity 0.0004° .
3. Integration Algorithm: Two levels higher accuracy than conventional DDA.
4. Multi-Transfer: 18 bit multi-transfer operation by 6 bit (multi-pass) multiplier.

5. . **Memory:** Delay line (or drum).
6. **Iteration Rate:** 500 it/sec with delay line (100 it/sec with drum) memory.
7. **Volume:** 1/3 cubic foot with delay line (1 cubic foot with drum) memory.

C. Airborne Strap-Down Computer (effects same computation as the Breadboard model now being fabricated).

1. **Performance:** Reference error - 7 sec, velocity error ≤ 0.1 ft/sec in 1 hour of airborne flight.
2. **Analogue to Digital Converter:** Whole-word sampler of angular rates, accelerometer inputs; granularity 0.0001 .
3. **Integration Algorithm:** Two levels higher accuracy than conventional DDA.
4. **Multi-Transfer:** 20 bit multi-transfer operation by 10 bit (multipass) multiplier.
5. **Memory:** Drum.
6. **Iteration Rate:** 130 it sec.
7. **Volume:** 1 cubic foot.

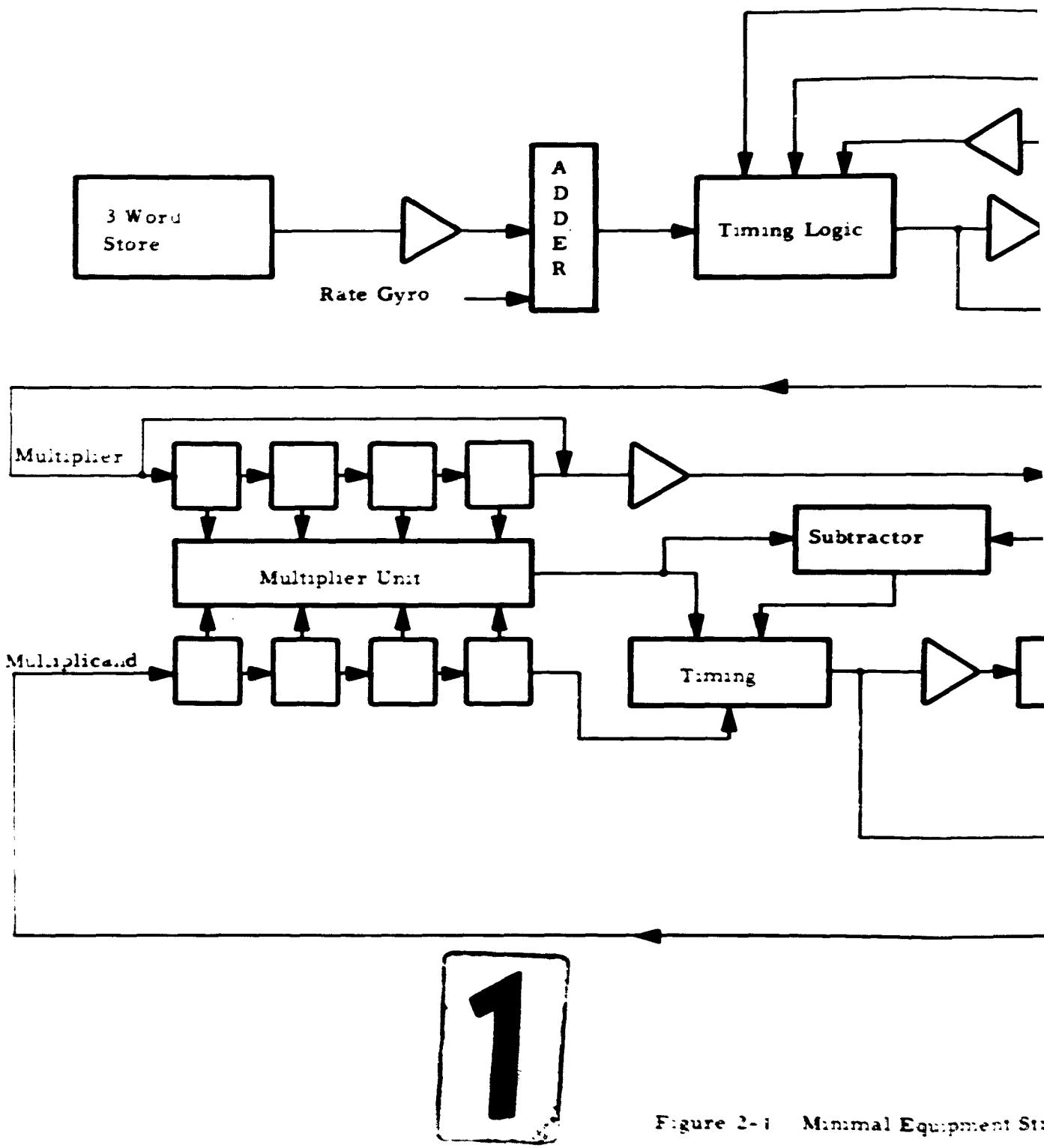


Figure 2-1 Minimal Equipment Schematic

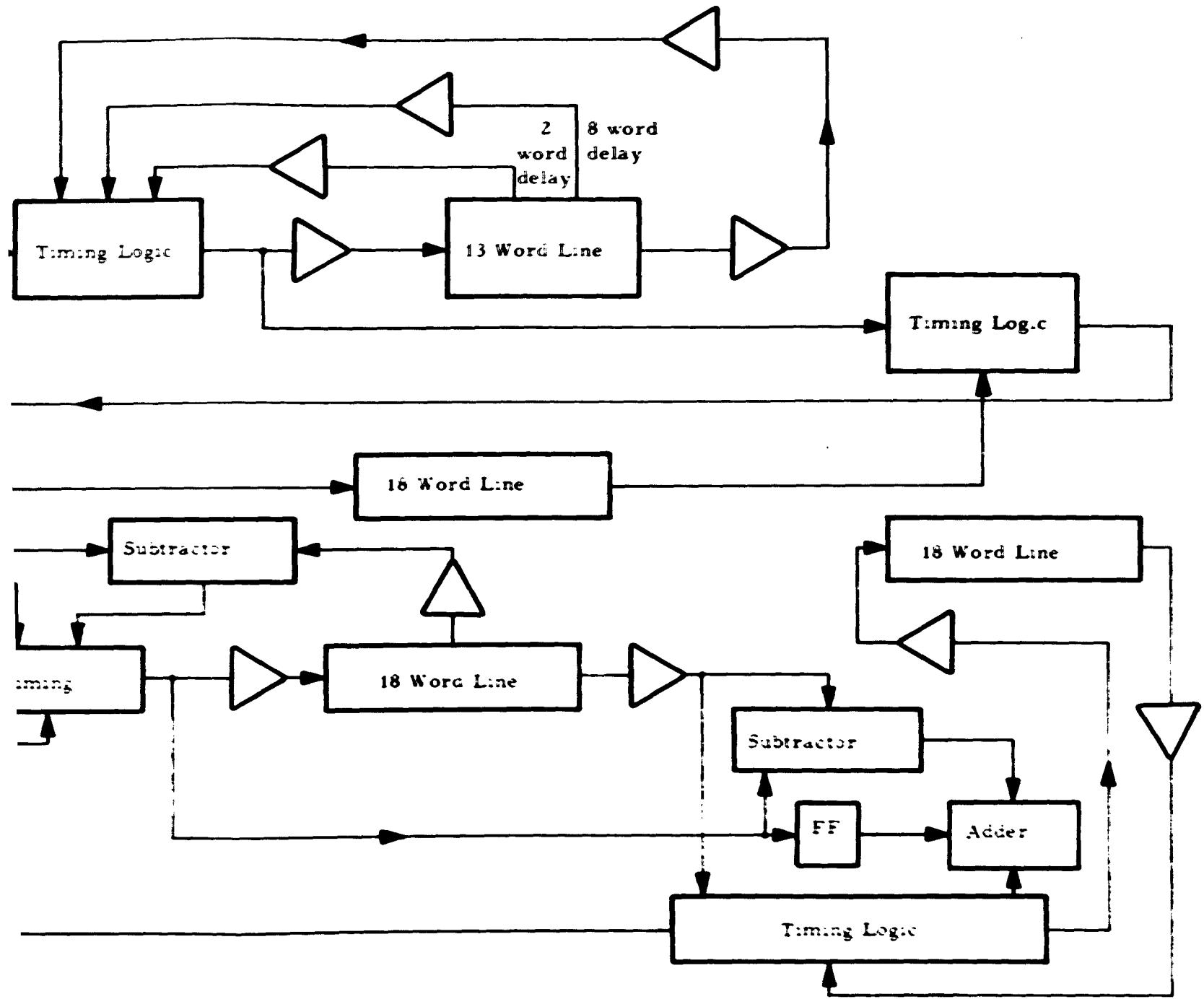


Figure 2-1 Minimal Equipment Strap-Down Computer

2.9 LOGICAL DESCRIPTION OF THE STRAP-DOWN PROCESSOR.

A. Introduction - The strap-down processor is a special purpose computer which solves the following equations:

$$\begin{aligned}\frac{dU_{j1}}{dt} &= w_3 U_{j2} - w_2 U_{j3} & j = 1, 2, 3 \\ \frac{dU_{j2}}{dt} &= w_1 U_{j3} - w_3 U_{j1} \\ \frac{dU_{j3}}{dt} &= w_2 U_{j1} - w_1 U_{j2} \\ \Delta V_j &= A_1 U_{j1} + A_2 U_{j2} + A_3 U_{j3}\end{aligned}\tag{II-27}$$

The angular velocities (w_1 , w_2 , w_3) and accelerations (A_1 , A_2 , A_3) are inputs to the computer from a magnetic tape. Each input has twenty bits including sign. The direction cosines which are computed are recorded on magnetic tape and are also routed back into the computer to be used for further computations. The velocities are recorded on the output tape but they are not used in the computer.

An iteration of the computer consists of the following cycles. The input data is transferred from the input tape into a core buffer memory unit. Then it is entered into the computer and operated on to solve the equations listed above. The results are then recorded on the output tape and also recirculated on the magnetic drum to be used in the following iterations.

The logical equations are expressed in a six-letter format. The unit designation (2nd letter) is as follows:

A	Input Accumulator
C	Control
E	Extrapolator
I	Tape, Core, and MCU inputs
M	Multiplier
N	Input
P	Control Panel
T	Output

The necessary reading, writing and control circuits are included in the computer to allow the magnetic tape units to be connected directly to the computer without any external control units. Either IBM 727 or IBM 729 II tape units can be used.

- B. Input Unit - The input unit consists of the tape input amplifiers, the tape-to-core buffer register, the core output buffer register, the serial input flip flops and two parity check circuits.
- Signals from the input tape are in an NRZI form, a change in magnetic flux indicating a binary "one". The signal is put into both an inverting amplifier and a non-inverting amplifier, and will alternately be read from these two circuits. Both circuits will not have a true output at the same time.
- The information on the tape is put into the tape-to-core buffer register (FNW00-FNW06). Because of skew on the tape, all seven bits of information might not be recorded on the same pulse, so as

soon as any bit in the tape-to-core buffer register is turned on the input counter begins to count from zero to seven. If the instruction register is set to "fill core," the information is written into the core on the count of five. If the instruction register is set to "ready for instruction" then the instruction register is set from the tape-to-core buffer register on the count of seven. The tape-to-core buffer register is reset on the count of seven.

Information is read from the core into the core buffer register under control of the input counter during the fill drum, set address and compute instructions. From the core output buffer register the information is shifted into the serial input flip flops and then into the input summing registers in the input accumulator unit. Each of the instructions will be explained in detail in another section.

Parity is checked (see Figure 2-2) both at the tape-to-core buffer register and the core output buffer. The tape check can stop the computer under control of the operator. This will be explained in more detail in another section. The core output parity check will turn on a light if an error is indicated but computer operation will not be interrupted.

- C. Input Accumulator Unit - The input accumulator (Figure 2-3) accepts accelerometer data and angular rate data from the serial input flip flops in the input unit. There are three accelerometer inputs (A_1^* , A_2^* , A_3^*) and three rate gyro inputs (W_1^* , W_2^* , W_3^*). During a compute cycle of 27 word times, which will be denoted by T , each input is summed into the input summing register once every third word time. If the first value is $W_i^*(t + T/9)$ or $A_i^*(t + T/9)$ then the final sum becomes

$$W_i^*(t + T/9) + W_i^*(t + 2T/9) + \dots + W_i^*(t + 8T/9) + W_i^*(t + T) = S(t + T)$$

(II-23)
II-29

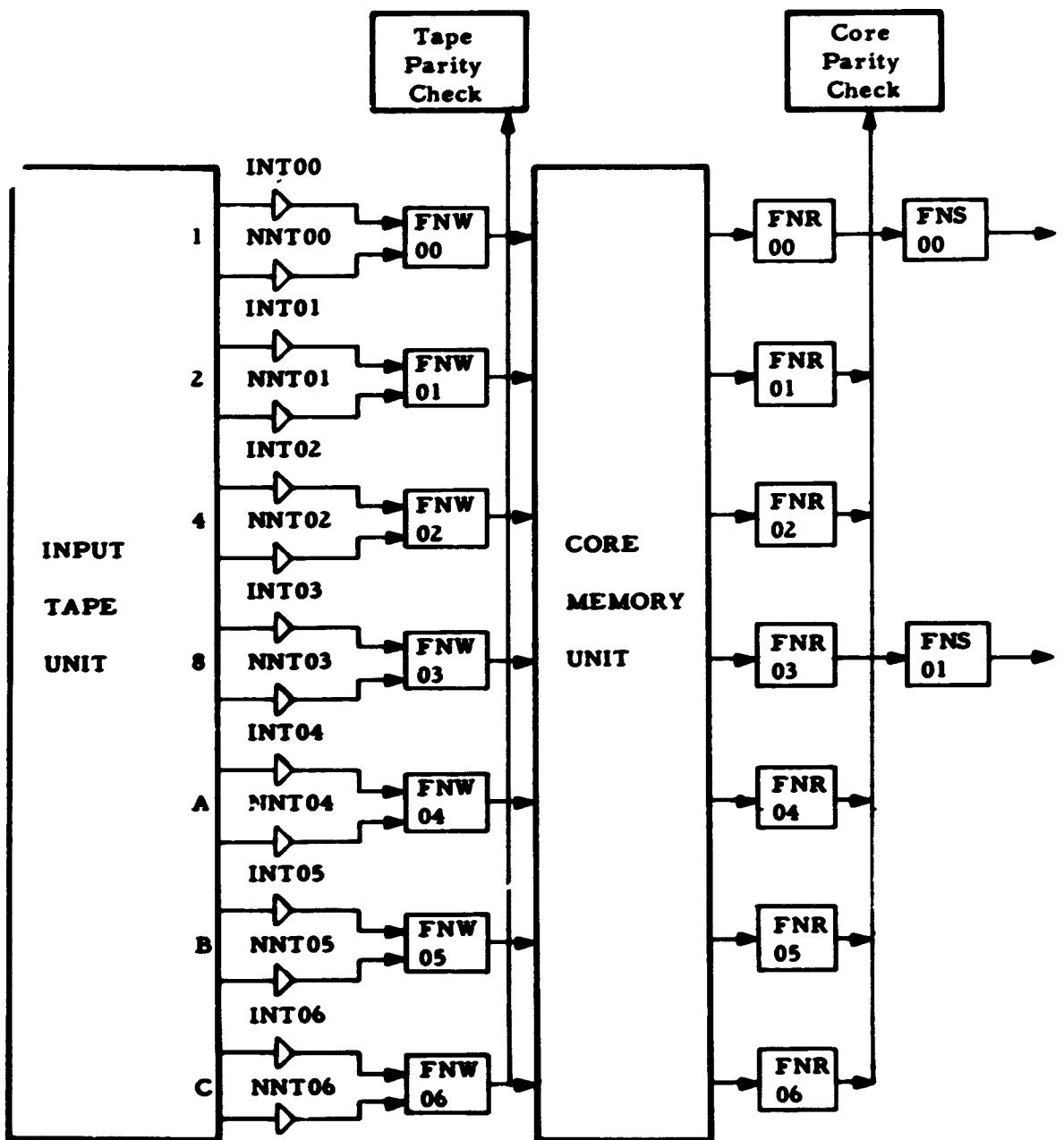


Figure 2-2. Input Unit

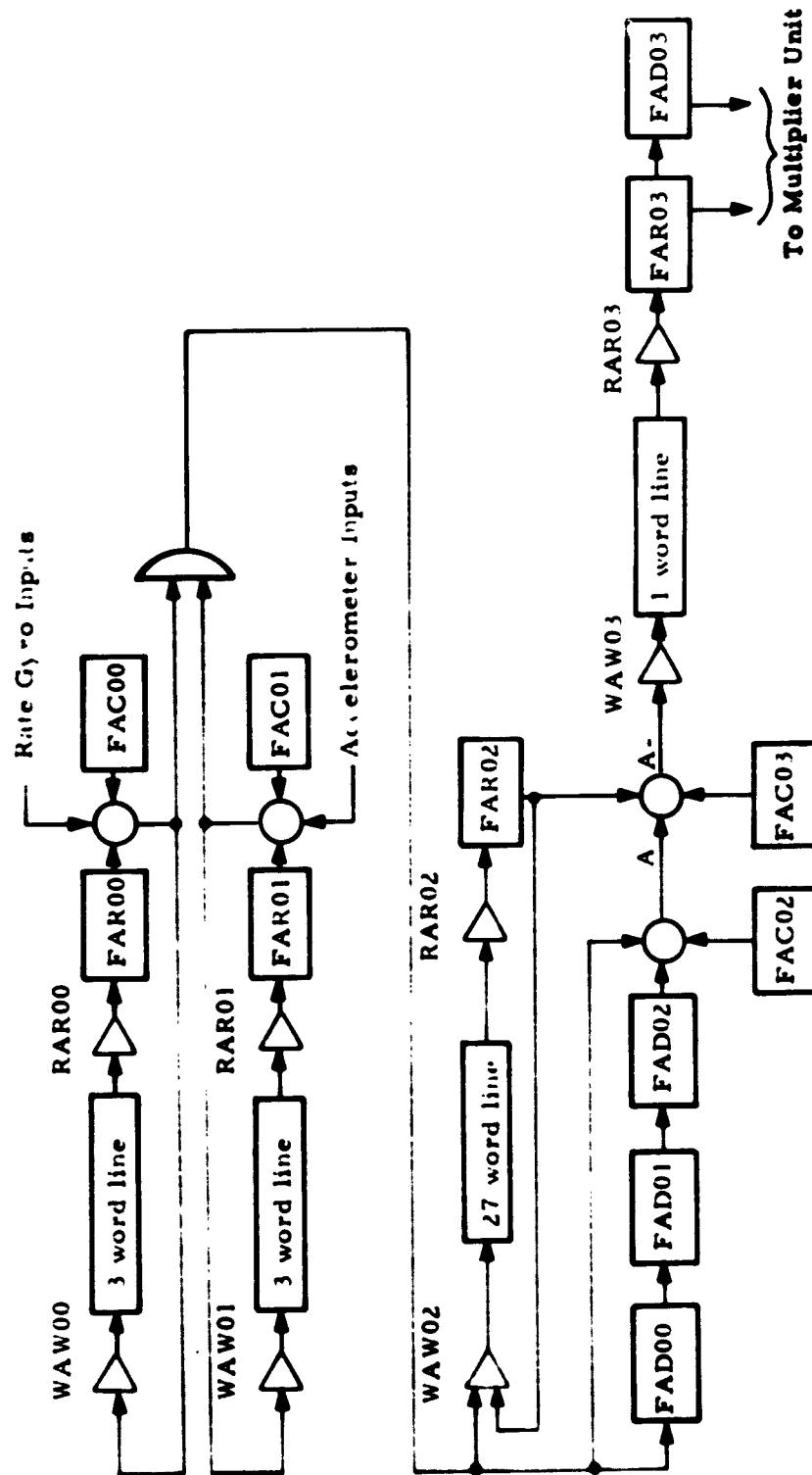


Figure 2-3. Input Accumulator Unit

The sum $S(t + T)$ goes to a 27 word drum line (WAW02), to a three-bit delay line (FAD00-FAD02) and into an adder (DAA02). The output of the 27-word line (FAR02) is $S(t)$, the output of the three-bit delay is $8S(t+T)$, and the output of DAA02 is $8S(t+T) + S(t+T) = 9S(t+T)$. FAR02 is subtracted from DAA02 in DAA03. This gives $9S(t+T) - S(t)$ which is W_i and A_i . These values are put into a one-word delay line (WAW03) at the proper word times and the output of this line is the multiplicand input to the multiplier unit. The flow of information is shown in Figure 2-4.

The computer word length is 32 bits and the input word length is 20 bits. The maximum length of $9S(t+T) - S(t)$ is 27 bits, and must be rounded off before being entered into the multiplier unit which has 20 bit registers.

The output of FAR03 is shown in Figure 2-5.

The round off is performed as follows. If bit 6 is zero enter bits 7-26 as they are. If bit 6 is a one and any of bits 0-5 are also a one, add one to bits 7-26. If bit 6 is a one and bits 0-5 are all zero then the multiplicand is increased by one only if bit 7 is a one.

- D. Multiplier Unit - The multiplier unit (Figure 2-5) consists of three twenty-bit registers, a twenty-bit parallel adder, a two's complement carry flip flop (FMC00), and a multiplier input flip flop (FMQ01). This unit multiplies a twenty-bit multiplicand by a thirty-two bit multiplier each word time. The output is a twenty-bit product which is placed in the least significant end of a thirty-two bit word. The multiplication process is basically that described by Booth and Booth.*

*Booth, A. D., and Booth, K. H. V., Automatic Digital Calculators, Academic Press, pp. 45-48, 1956.

WORD	DAA00	DAA01	DAA02	FAR03
0				
1				
2	W2		W2	W2
3				W2
4				W2
5				W2
6				W2
7				
8				
9	W3		W3	W3
10				W3
11				W3
12				
13				
14				
15				
16		A3	A3	A3
17		A2	A2	A2
18				A2
19				A2
20				A2
21		A1	A1	A1
22				A1
23				A1
24				
25	W1		W1	W1
26				

Figure 2-4. Information Flow in Input Accumulator Unit

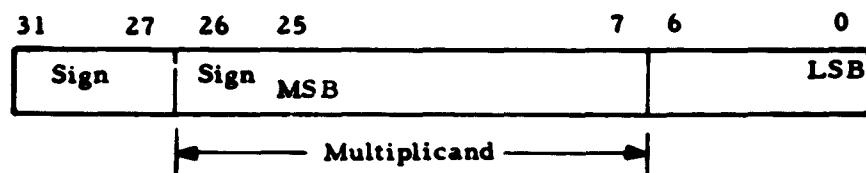


Figure 2-5. Output of FAR 03.

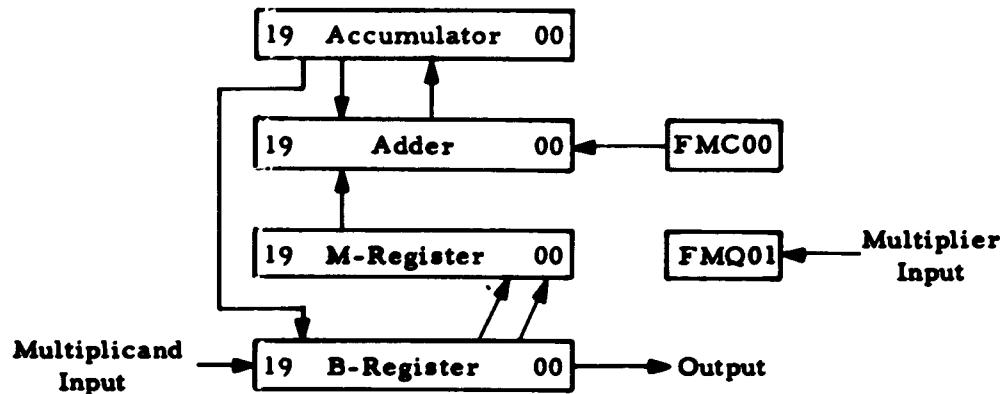


Figure 2-6. Multiplier Unit

The multiplicand input is entered serially into the B-register from the input accumulator unit during bit times 7-25. During bit times 26-30 the B-register remains static with the first 19 bits of the new multiplicand in bits 1-19 and the sign of the last product is bit zero. This allows the output to be in the form shown below in Figure 2-7.

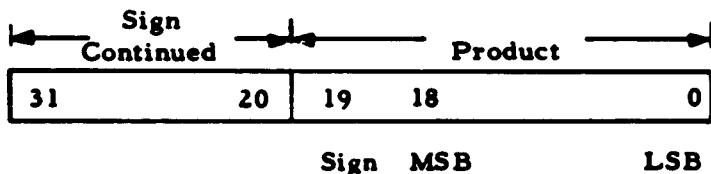


Figure 2-7. Multiplier Output

At bit time 31 the complement of the sign bit of the multiplicand is entered into FMM19 and the complement of the B-register is shifted into the M-register shifted right one bit. This puts the complement of the multiplicand into the M-register and the multiplier unit is ready for the next multiplication. As is shown in Figure 2-8, the multiplicand does not change every word time. The new multiplicand

Word Time	Multiplicand	Multiplier
0	w_1	U_{13}
1	w_1	U_{23}
2	w_1	U_{33}
3	w_1	U_{12}
4	w_1	U_{22}
5	w_1	U_{32}
6	w_2	U_{11}
7	w_2	U_{21}
8	w_2	U_{31}
9	w_2	U_{13}
10	w_2	U_{23}
11	w_2	U_{33}
12	w_3	U_{12}
13	w_3	U_{22}
14	w_3	U_{32}
15	w_3	U_{11}
16	w_3	U_{21}
17	w_3	U_{31}
18	A_3	U_{13}
19	A_3	U_{23}
20	A_3	U_{33}
21	A_2	U_{12}
22	A_2	U_{22}
23	A_2	U_{32}
24	A_1	U_{11}
25	A_1	U_{21}
26	A_1	U_{31}

Figure 2-8 . Multiplication Schedule

is brought in as described above only during word times 5, 11, 17, 20, 23 and 26. During all other word times the multiplicand is already in the M-register, either in true or complemented form. If it is in complemented form, the two's complement flip flop (FMC00) will be on. If it is not in this form then the M-register is complemented and FMC00 is turned on at bit time 31. The reason for this is that in the Booth and Booth multiplication, a subtraction is the first operation.

The multiplier inputs come from two read heads on a drum line in the output unit. The correct value is at FTR05 during word times 0-17 and at FTR06 during word times 0-2 and 6-26. It is read from FTR05 during word times 0-17 and from FTR06 during word times 6-26.

The contents of the output read flip-flop (FTR05 or FTR06) and FMQ01 control the operation of the multiplier unit. If an addition or subtraction is to be done, the adder output is transferred to the accumulator on the half clock. Then the accumulator is shifted right on the master clock. The sign bit is shifted into bit 18 and the sign remains unchanged at all bit times except bit time 11. During this bit time, "one" is effectively added to bit 18 by shifting the complement of the sign into 18, and resetting the sign to zero. Since twenty shifts remain, the added "one" at bit time 11 has the effect of rounding off the product.

- E. **Extrapolator Unit** - The output of the multiplier unit goes to the extrapolator unit (Figure 2-^a) where it goes through an extrapolation by T/2 and a multiplication by 12. If the input to the extrapolator unit is $y(t)$ then the output is:

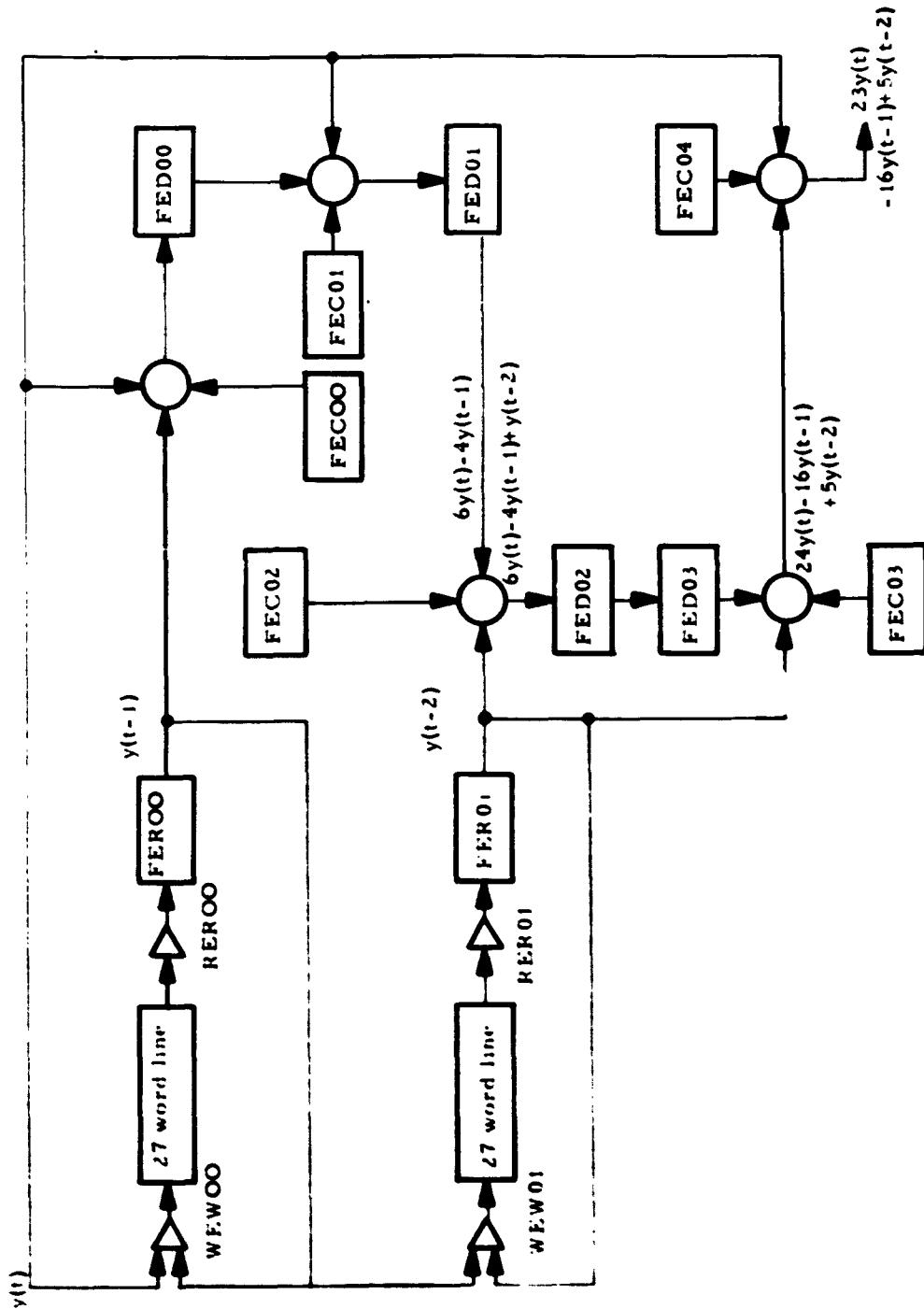


Figure 2-9 Extrapolator Unit

$$12 \left[y(t) + 1/2\Delta y(t) + 5/12\Delta^2 y(t) \right] = 23y(t) - 16y(t-1) + 5y(t-2) \quad (\text{II-29})$$

The multiplication by 12 is included to avoid a division by 12. Figure 2-9 shows the mechanization of the extrapolator unit.

F. Output Unit- The output unit (Figure 2-10) accepts inputs from DEA04 and processes them on three drum lines as shown in the schedule of Figure 2-11. The output from DEA04 is a multiplication of two variables and these must be combined to form the following equations:

$$\begin{aligned} \frac{dU_{j1}}{dt} &= W_3 U_{j2} - W_2 U_{j3} & j &= 1, 2, 3 \\ \frac{dU_{j2}}{dt} &= W_1 U_{j3} - W_3 U_{j1} \\ \frac{dU_{j3}}{dt} &= W_2 U_{j1} - W_1 U_{j2} & (\text{II-30}) \\ V_j &= A_1 U_{j1} + A_2 U_{j2} + A_3 U_{j3} \end{aligned}$$

After the above equations have been formed, they must be available for use as a multiplier input (the direction cosines only) at the correct word times, and they must be recorded on the output tape.

The direction cosines must be fed back into the multiplier unit in the order indicated in the description of the multiplier unit. The correct variable is at FTR05 during word times zero through 17 and at FTR06 during word times six through 26.

The information that is recorded on the output tape is first put on the one-word drum line through WTW02 where it is recirculated for at least three word times. The drum line inputs are shown in Figure 2-12. The tape accepts a character of

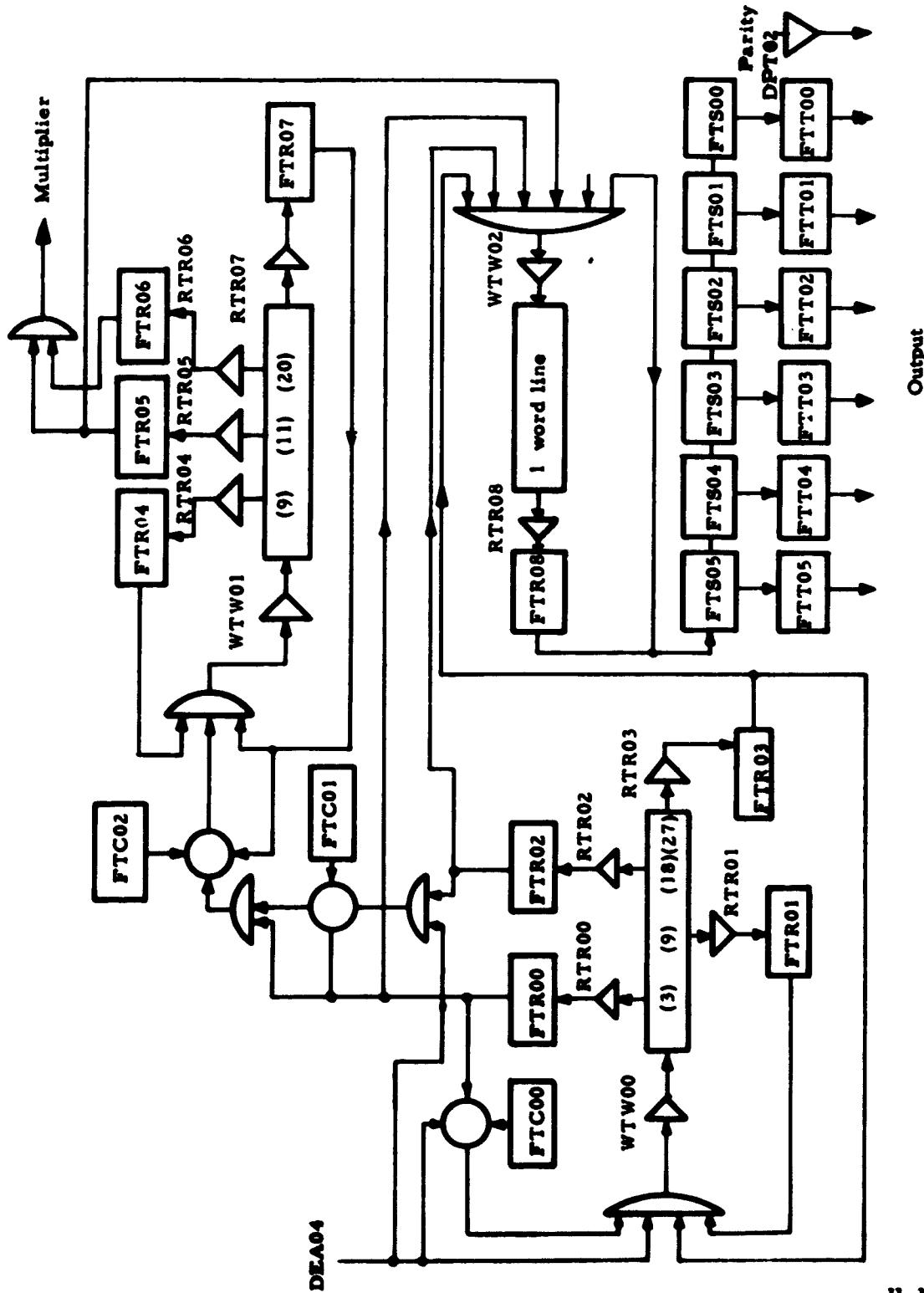


Figure 2-10. Output Unit

Word	DLEA 64	(To-3) FTR00	(To-9) FTR01	(To-16) FTR02	(To-27) FTR03	(T1-9) FTR04	(T1-11) FTR05	(T1-20) FTR06	(T1-27) FTR07	FTR08*	FTR08*
0	A1U31 A3U32 A2U32	W3U31	ΔV_3		W1U13	W1U13	U33	U13	U33	Address	U31
1	W1U13 ΔV_1	A3U13			W2U13	W1U23	U12	U23	U12		
2	W1U23 ΔV_2	A3U23			W2U23	W1U33	U22	U33	U22		
3	W1U33 ΔV_3	A3U33			W2U33	W1U33	U32	U12	V1		
4	W1U12 W1U13	A3U13+A2U12			W2U13	W1U12	U11	U22	V2	ΔV_1	
5	W1U22 W1U23	A3U23+A2U22			W2U23	W1U22	U21	U32	V3	ΔV_2	
6	W1U32 W1U33	A3U33+A2U32			W2U33	W1U32	U31	U11	U11	ΔV_3	
7	W2U11 W1U12	ΔV_1			W3U11	ΔV_1	U13	U21	U21		
8	W2U21 W1U22	ΔV_2			W3U21	ΔV_2	U23	U31	U31		
9	W2U31 W1U32	ΔV_3			W3U31	ΔV_3	U33	U13	U33		
10	W2U13 ΔV_1	W1U13			A3U13	W2U13	U12	U23	V1		
11	W2U23 ΔV_2	W1U23			A3U23	W2U23	U22	U33	V2	ΔV_1	
12	W2U33 ΔV_3	W1U33			A3U33	W2U33	U32	U12	V3		
13	W3U12 W2U13	W1U12	A3U13+A2U12		W2U13	W2U13	U11	U22	U22		
14	W3U22 W2U23	W1U22	A3U23+A2U22		W2U23	W2U23	U21	U32	U21		
15	W3U32 W2U33	W1U32	A3U33+A2U32		W2U33	W2U33	U31	U11	U31		
16	W3U11 W2U13	ΔV_1			W3U11	ΔV_1	U13	U21	U13		
17	W3U21 W2U23	ΔV_2			W3U21	ΔV_2	U23	U31	U23		
18	W3U31 W2U33	ΔV_3			W3U31	ΔV_3	U33	*U13	U33	ΔV_2	
19	A3U13 W3U11	W2U13			W1U13	A3U13	V1	*U23	U23		
20	A3U23 W3U21	W2U23			W1U23	A3U23	V2	*U33	U33		
21	A3U33 W3U31	W2U33			W1U33	A3U33	V3	V1	U12		
22	A2U12 A3U13	W2U13			W1U12	A3U13+A2U12	*U11	V2	U22		
23	A2U22 A3U23	W2U23			W1U22	A3U23+A2U22	*U21	V3	U32		
24	A2U32 A3U33	W2U33			W1U32	A3U33+A2U32	*U31	*U11	U11		
25	A1U11 A3U3+A2U12	W3U11	ΔV_1		A3U13	ΔV_1	*U13	*U21	U13		
26	A1U21 A3U23+A2U22	W3U21	ΔV_2		A3U23	ΔV_2	*U23	*U31	U31		

*Result of present iteration
**Compute cycle
see last 27 words after compute cycle

Figure 2-11. Information Flow in Output Unit

Word	WTW00	WTW01	WTW02*	WTW02**
0	DEA04+FTR00	FTR04	FTR08	FTR08
1	DEA04			
2				
3				
4				
5				
6				
7	FTR01	DEA04-FTR00+FTR07		
8	FTR01			
9	FTR01			
10	DEA04	FTR00+FTR07		
11	DEA04			
12	DEA04			
13	FTR00	DEA04-FTR00+FTR07		
14				
15				
16	DEA04	FTR04		
17				
18				
19		FTR02-FTR00+FTR07		
20				
21				
22	DEA04+FTR00	FTR04		
23				
24				
25				
26				

*Compute
**last 27 words after compute

Figure 2-12. Drum Line Inputs

seven bits in parallel at the rate of two characters per word time. A 32 bit word is divided into six characters each containing six bits of information and one parity bit, and it is recorded on the tape during three consecutive word times. Four of the bits are recorded twice. For each iteration there are 13 words of information to be recorded on the output tape: one address, 9 direction cosines, and three velocity increments. This is done within the compute cycle and the first 27 words after the compute cycle.

During each of these 54 word times a character is recorded at bit time four and bit time 20. The order in which the information is output can be seen in Figure 2-11 as the output of FTR08.

The output of the one-word drum line, FTR08, is continuously being transferred into FTS05-FTS00 as shown in Figure 2-10. Then, at the proper times, FTS00-FTS05 are transferred in parallel into FTT00-FTT05. A parity bit is generated in DTP02 and then the character is recorded on the tape either at bit time four or bit time 20. Figure 2-13 shows the organization of the output word. Note that bits 5, 10, 21 and 26 are each output twice. This fills out the 66 bits of information, in the six characters.

FTS(n)		FTT(n)		Output Pulse		Output Word
Bit Time	Word Time	Bit Time	Word Time		Bits	
6	n	20-21	n		0-5	
0	n+1	4-5	n+1		26-31	
16	n+1	20-21	n+1		10-15	
22	n+1	4-5	n+2		16-21	
11	n+2	20-21	n+2		5-10	
27	n+2	4-5	n+3		21-26	

Figure 2-13. Output Organization

G. Control Unit - The control unit is made up of a number of flip flops which are used to control the cycles in all the other units, the amplifiers which specify certain conditions of the flip flops, and the clock system.

- 1. Bit Counter (FCB00-FCB04)** - The bit counter is a binary counter which counts from 0 to 31 stepping one count on every master clock pulse. The states of the counter are shown in Figure 2-14.

State	FCB04	FCB03	FCB02	FCB01	FCB00
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0
31	1	1	1	1	1

Figure 2-14. Bit Counter

2. Word Counter (FCW00-FCW05) - The word counter counts from 0 to 26 as shown in Figure 2-15. It counts up one state every time the bit counter is in state 31.

	FCW05	FCW04	FCW03	FCW02	FCW01	FCW00
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	1	0	0
4	0	0	0	1	0	1
5	0	0	0	1	1	0
6	0	0	1	0	0	0
7	0	0	1	0	0	1
8	0	0	1	0	1	0
9	0	1	0	0	0	0
10	0	1	0	0	0	1
11	0	1	0	0	1	0
12	0	1	0	1	0	0
13	0	1	0	1	0	1
14	0	1	0	1	1	0
15	0	1	1	0	0	0
16	0	1	1	0	0	1
17	0	1	1	0	1	0
18	1	0	0	0	0	0
19	1	0	0	0	0	1
20	1	0	0	0	1	0
21	1	0	0	1	0	0
22	1	0	0	1	1	1
23	1	0	0	1	0	0
24	1	0	1	0	0	1
25	1	0	1	0	0	0
26	1	0	1	0	1	0

Figure 2-15. Word Counter

3. **Input Counter (FCN00-FCN02)** - The input counter is used to control the timing of all input operations, the resetting of the instruction register at the end of the compute cycle, and the resetting of the output counter at the beginning of an output cycle. The count sequences are shown in the timing diagrams.
4. **Instruction Register (FCC00-FCC02)** - The instruction register (Figure 2-16) controls what the computer is doing. When it is in state zero, ready for instruction, then an instruction will be set into this register from the input tape. This will cause the computer to go through a cycle of one or two instructions and then return to the "Ready for Instruction" state.

State	Instruction	FCC02	FCC01	FCC00
0	Ready for Instruction	0	0	0
1	Fill Core	0	0	1
2	Start Output Tape	0	1	0
3	Stop Tapes	0	1	1
4	Fill Drum	1	0	0
5	Prepare to Fill Drum	1	0	1
6	Set Address	1	1	0
7	Compute	1	1	1

Figure 2-16. Instruction Register

5. **Ferranti Flip Flop (FCC03)** - This flip flop is turned on with the master clock and turned off with the half clock. It is used to generate a Ferranti write pattern for the write amplifiers.

6. Start Control - What is needed to start the computer properly is a signal that is true for one bit time only. When the start button is depressed FCC04 is turned on. Then, when the bit counter is in state 31, the start signal (NCC20 and ICC20) is true. This start signal starts the input tape and resets the instruction register, the tape-to-core buffer register and the parity check flip flops. The computer then idles until it receives an instruction from the input tape.

Before the computer is started the clear core button must be depressed. This will load the core with zeros and set the address register to word zero. To prevent tapes from starting prematurely the following sequence of operations must be performed in the correct order when turning on the computer power.

- (a) Press the reset button on both tape units.
- (b) Turn on the power.
- (c) Press the halt buttons.
- (d) Press the start button on both tape units.

7. Tape Motion Control

a. Input Tape - The motion of the input tape is controlled by FCN05. When it is turned on the tape will start and when it is turned off the tape will stop. This tape is always started with the start button. The following conditions will stop the tape:

- (1) The computer accepts a "stop tapes" instruction from the tape.
- (2) The halt buttons are depressed.

- (3) A tape parity error is detected and the parity control switch is set to halt.
 - (4) The tape has backed up to reread a record.
 - (5) The computer is in the single cycle mode and the core is filled.
- b. Output Tape - The motion of the output tape is controlled by FCT00. When it is turned on the tape will start and when it is turned off the tape will stop. The tape will start when the computer accepts a "start output tape" instruction from the input tape or when the start button is depressed and the tape parity check flip flop (FCN03) is on. The following conditions will stop the tape:
- (1) The computer accepts a "stop tapes" instruction from the input tape.
 - (2) The halt buttons are depressed.
 - (3) A tape parity error is detected and the parity control switch is set to halt.
 - (4) The computer is in the single cycle mode and the core is filled.

At the beginning of a problem, the output tape is started under control of the input tape to allow a 3.5 inch file gap to be placed on the output tape. To do this the tape parity check flip flop must be off when the start button is depressed. If the tape parity check light is on, then the reset error button must be depressed before the computer is started. A tape parity error will stop both tapes. This will prevent running out a lot of output tape while the input tape is being backed up. When the

computer is again started, both tapes must start simultaneously so the parity check circuits must not be reset then. The start button will reset the tape parity check flip flop.

8. Error Control

- a. Tape Error - During the "Ready For Instruction" and "Fill Core" modes of the computer operation, information is transferred from the input tape to the instruction register and the core buffer unit. All circuits not directly involved in this transfer of information are idling, thus if a parity error is detected at this time, the record in which the error occurred can be reread. The parity is checked at the tape-to-core buffer register and if it is incorrect FCN03 will be turned on and the tape parity light will be turned on. If the parity control switch is set to halt, the tapes will stop. The record can then be reread as follows:
 - (1) Set the tape direction control switch to reverse and press the start button. This will back up the input tape to the beginning of the record.
 - (2) Press the clear core button. This will prepare the core to be filled from word zero.
 - (3) Set the tape direction control to forward and press the start button. This will start both tapes and reset the parity check flip flop.

The reset parity button must not be depressed as this will prevent the output tape from starting. If the parity is reset then it can be turned on again with the test parity button, and must be done before the computer is started.

If the parity control switch is set to ignore then the tape parity light will turn on but the computer will not stop.

- b. **Core Error** - During the "Fill Drum", "Set Address", and "Compute" instructions parity is checked at the core buffer register. The register is set with PCX06 and immediately starts shifting, so the parity must be checked the bit time after it is set.

PCX06 turns on FCN06 which turns itself off the following bit time. The core parity flip flop (FCN04) will be turned on if FCC02 (correct instruction), and FCN06 are both on and there is an even parity. FCN03 turns on the core parity light, but the computer continues to run. The light can be turned off with the reset parity button.

9. **Output Write Control** - Three flip flops control the write cycle to the output tape. FCT01 inhibits the tape write pulse when it is on. FCT02 and FCT03 control the lengths of the record gap, output data and output excess cycles. This is shown in Figure 2-17. Information is transferred from the output register, FTT00-FTT05 and DTP02, through their line drivers onto the output tape whenever there is a tape write pulse (LCX36). This is an eight microsecond pulse during bits 4-5 and 20-21 when the IBM 727 tape unit is used. The IBM 729 tape unit requires four microsecond pulses and they will occur at bit time four and bit time twenty. The write check character pulse tells the tape unit to record the longitudinal parity character. For the IBM 727 tape unit this is a 16 microsecond pulse (LCX37), and for the IBM 729 tape unit the signal is the turn off of a flip flop (FCT04) which was turned on by the first character of the record. The check character is recorded four character spaces after the last character of the record.

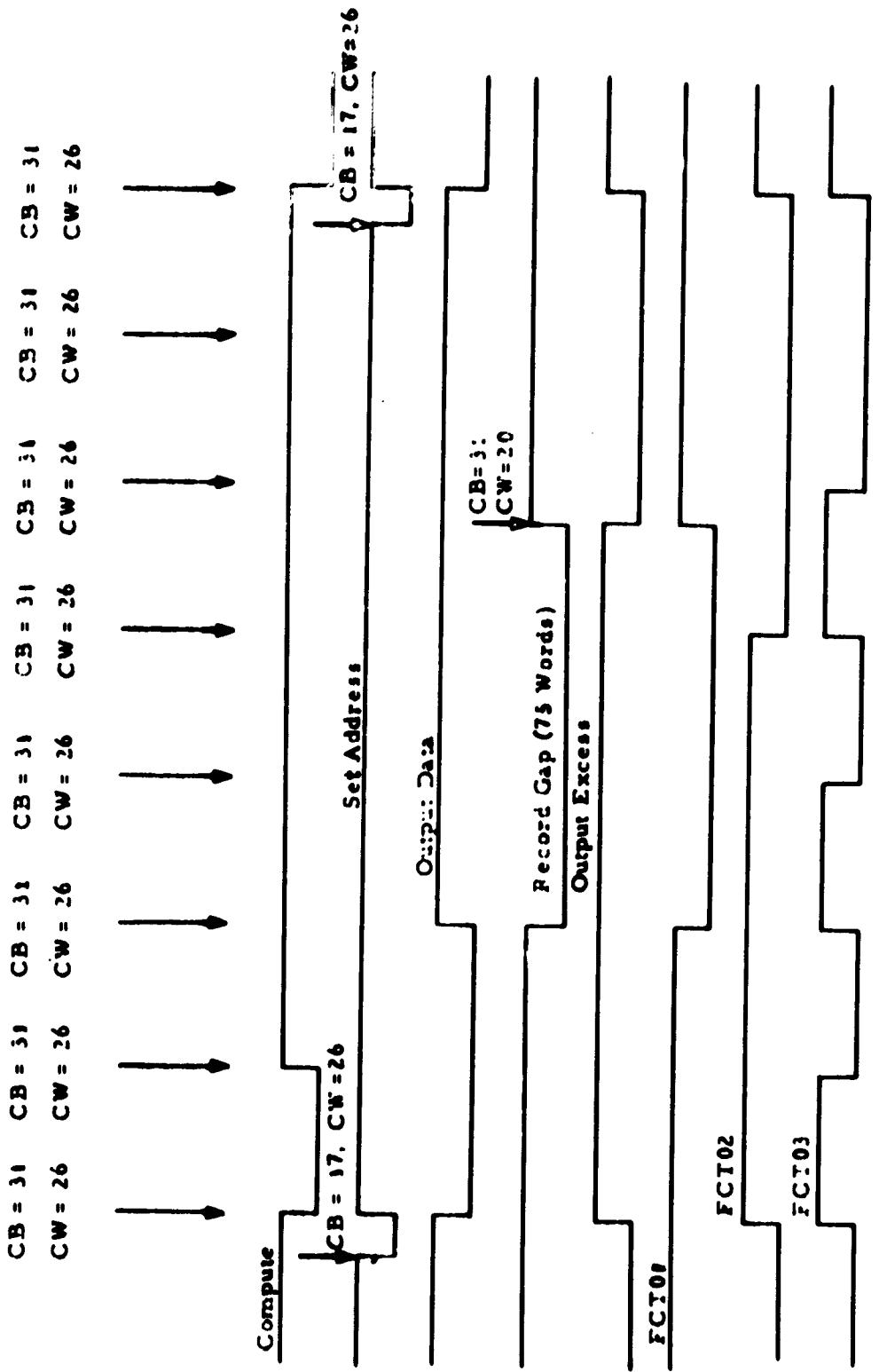


Figure 2-17. Output Cycle

H. COMMANDS

1. Ready for Instruction - The computer accepts the next character on the input tape as an instruction, providing the character has a zero in channel eight. The rest of the computer idles. See Figure 2-18.
2. Fill Core - The information on the input tape is routed through the tape-to-core buffer register into the core buffer unit. The core will accept 193 characters and then signal the computer that it is filled. The instruction register will then return to a ready for instruction configuration. See Figure 2-19.
3. Start Output Tape - When the instruction register is set to "Start Output Tape," the output tape motion control flip flop (FCT00) is turned on and the output tape write control flip flop (FCT01) is turned on, starting the tape and inhibiting the write pulse. The output counter starts its cycle when the tape mark is recognized on the input tape. The computer will accept instructions from the input tape when the instruction register is in this configuration.
4. Stop Tapes - This instruction indicates the end of a tape. It stops both tape units and turns on the end of tape light on the control panel.
5. Fill Drum - The instruction on the tape is "Prepare To Fill Drum." This instruction synchronizes the input counter to the bit counter and word counter for the fill drum operation. At bit 24 of word 26 the input counter is set to three. Then on bit 31 and with the input counter again at three, the instruction register changes from prepare to fill drum to fill drum. The drum lines now change from individual recirculation to a serial connection as shown in Figure 2-20. A total of 144 characters are put on the drum from the core

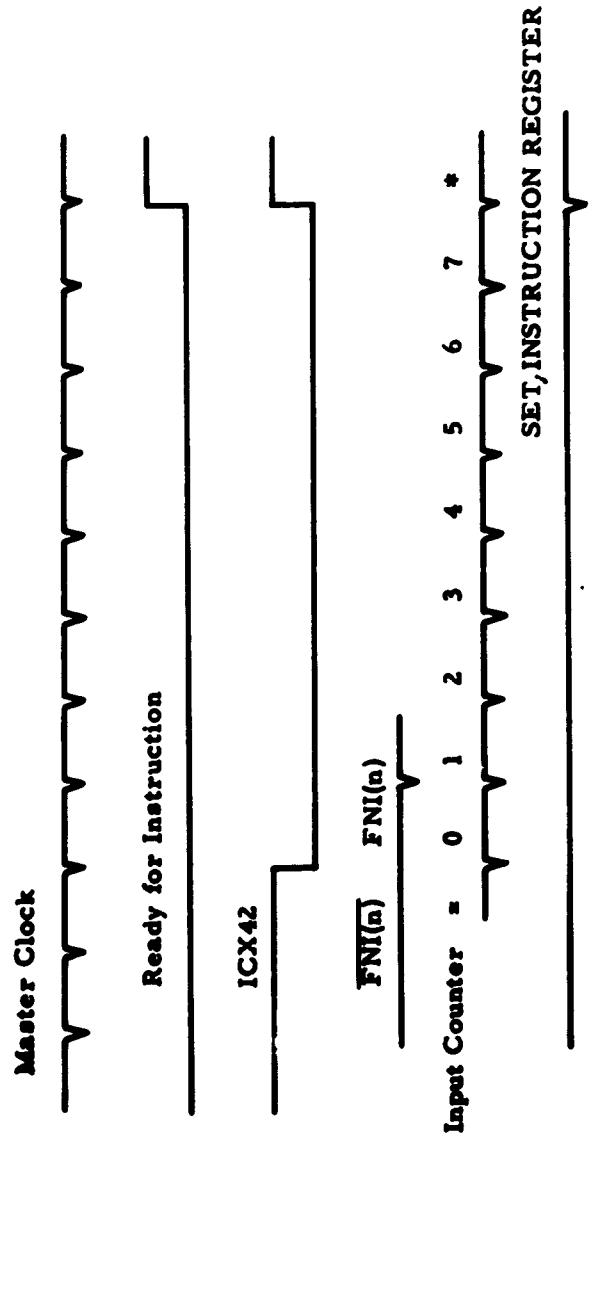


Figure 2-18. Ready for Instruction

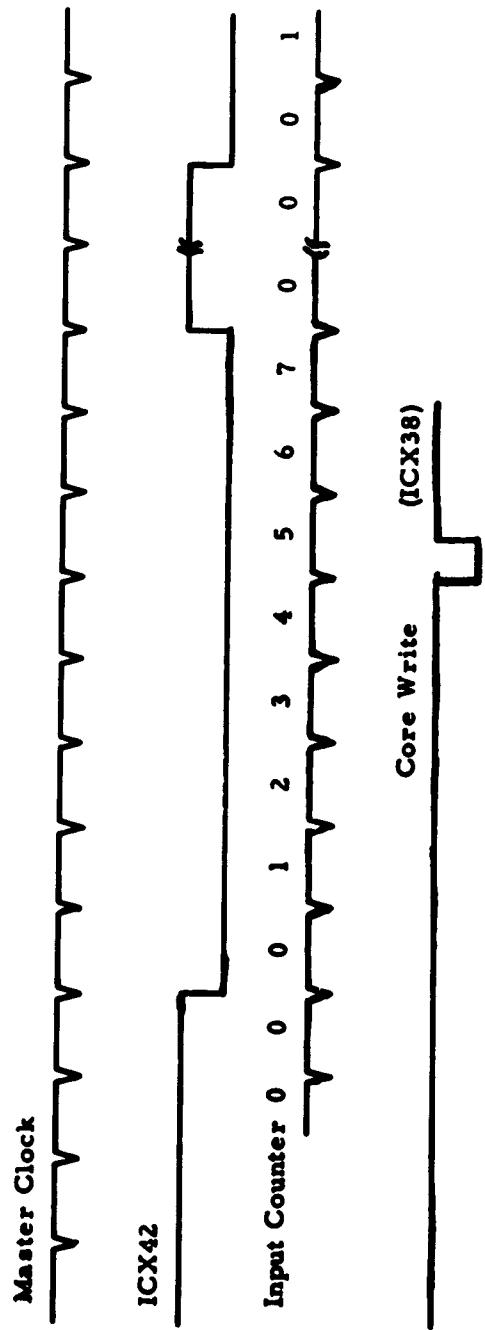


Figure 2-19. Fill Core

buffer unit. This is the amount of information on a 27-word drum line. Six fill drum instruction cycles are needed to completely fill the drum. See Figure 2-21.

6. Set Address - The set address instruction transfers the address of an iteration from the core buffer unit to the one-word output line. This consists of four characters and upon completion the instruction register changes to the compute configuration. See Figure 2-22.
7. Compute - The compute instruction is started automatically at the end of the set address instruction. During this cycle of 27 word times the data is read into the input summing registers and sent through the computer. The instruction register returns to the ready for instruction configuration at the end of the cycle. See Figure 2-23.

I. TAPE FORMAT

1. Input Tape - The tape must have a 3.5 inch file gap before the beginning of any recorded information. At the end of the file gap is a tape mark and its associated check character.

During word times 0-5 of a compute cycle, the multiplicand register holds W1. This is entered into the register during the last word of the previous compute cycle, so the register must be preset before the first cycle. This is accomplished by filling the input accumulator drum lines and going through a dummy compute cycle. The correct value will be entered into the multiplicand register if -W1 is filled on the drum to read from FAR02 at word time 25, and the data during the dummy compute cycle is all zeros. This is shown in Figure 2-24.

After the multiplicand register is preset the drum must be filled with the initial conditions. This requires six records, each record having enough information to fill a twenty-seven word line. The record consists of a fill core character, followed by 144 data characters, 49 excess characters which are needed to completely fill the core buffer unit but are not entered into the computer, three excess characters which cannot look like instructions, and finally a fill drum character. The last three excess characters are necessary to fill the record out to a multiple of six characters which is an IBM word. The drum line interconnections during the fill drum mode is shown in Figure 2-20. The layout of a fill drum record is shown in Figure 2-25.

After the drum is filled, the output tape must be started. A six-character record is used to do this and it is shown in Figure 2-26. Following this short record is a file gap instead of the normal record gap. This is used to provide the output tape with a file gap. The computer is now ready to accept input data and compute. One record of 198 characters as shown in Figure 2-26 is used for each iteration. The first character is a fill core instruction. This is followed by 193 characters of data, ordered as shown in Figure 2-27. After the data the set address instruction is used to start the compute cycle. Three excess characters are used between the last data character and the set address character so that the record will consist of an integral number of IBM words. The first character of the last record is a stop tape instruction. This will stop both tapes and turn on the end of tape light.

2. Output Tape - The output tape is started with an instruction on the input tape, and then nothing is written on the tape until the tape mark on the input tape starts the write cycle. No tape mark is recorded on the output tape.

The record consists of excess information followed by the output data. The excess information is zeros in channels 1, 2, 4, 8, A and B, and a one in channel C. The last three characters of the excess information will have a one in channel B and zeros in the other channels. The output data is a total of 108 characters in length, but only 76 characters contain output information. The output record is shown in Figure 2-28 and Figure 2-29.

J. LABORATORY MODELS OF HSDDA - Photographs of the laboratory models of the HSDDA Computer and the manual control unit for the computer are reproduced as Figures 2-30 and 2-31.

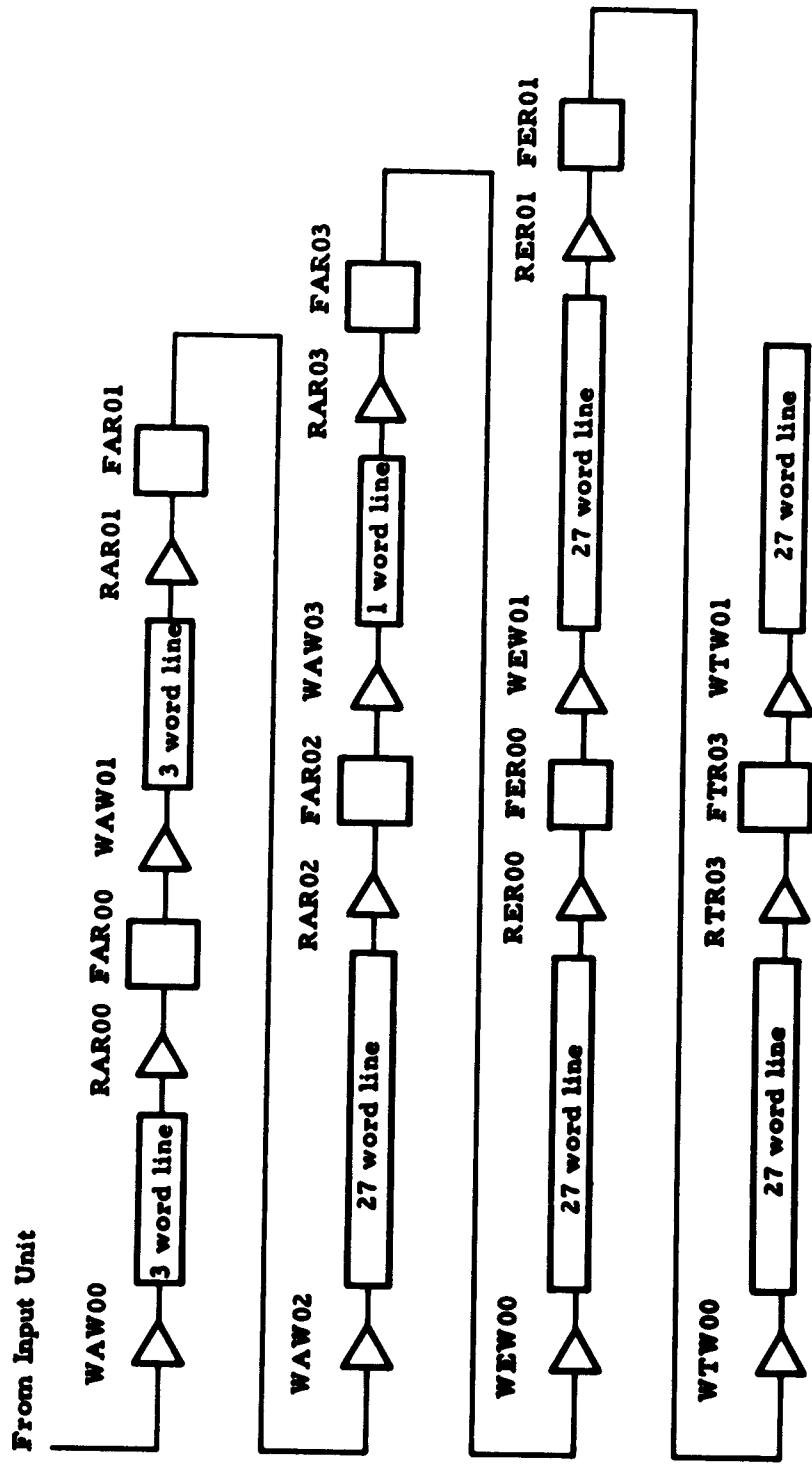


Figure 2-20. Serial Drum Connections for Drum Fill.

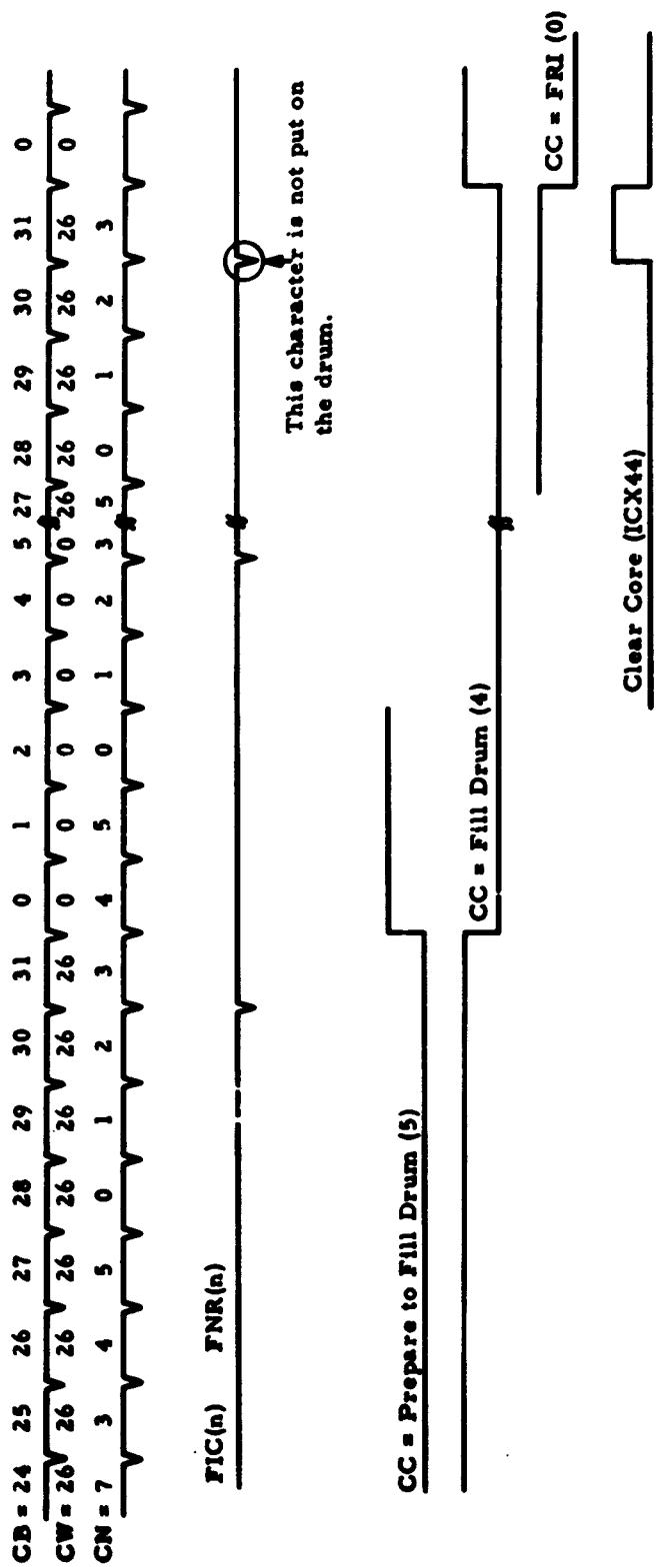
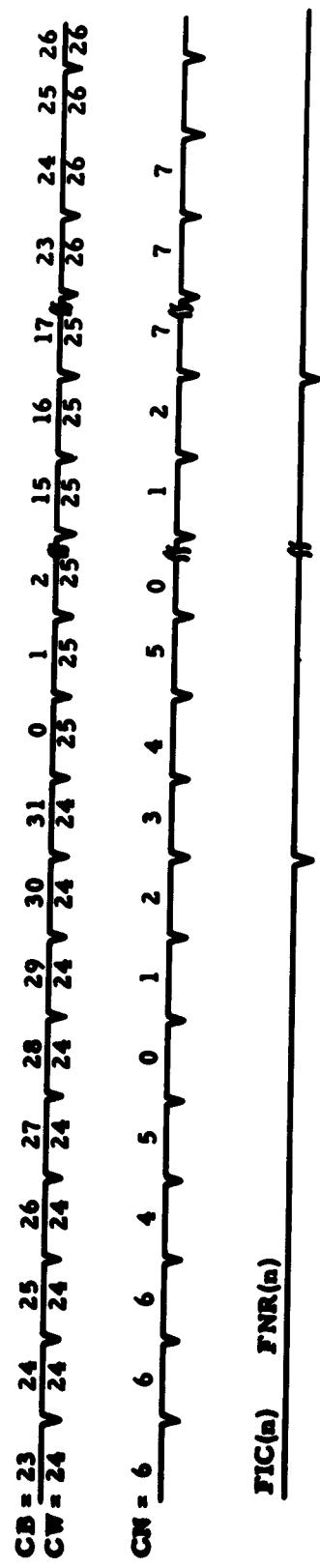


Figure 2-21. Fill Drum

Figure 2-22. Set Address



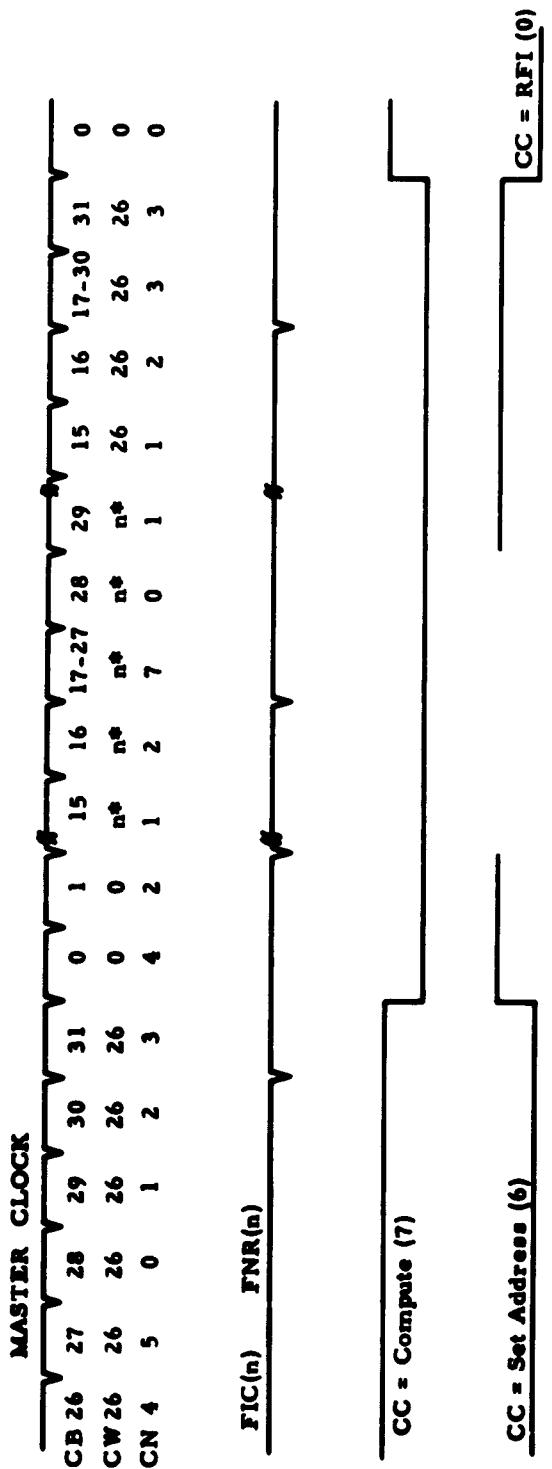


Figure 2-23. Computer

$\Phi_n = 26$

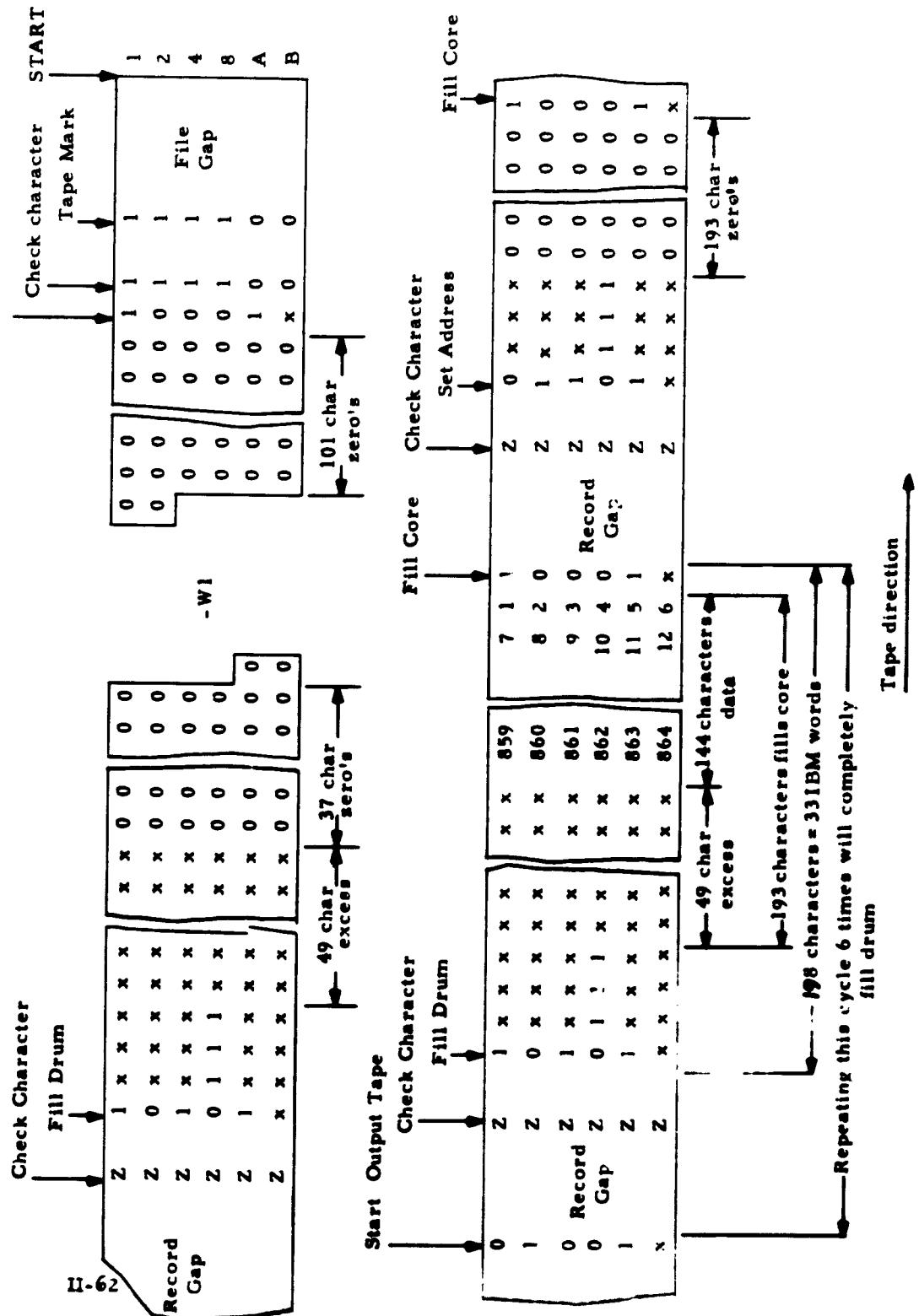


Figure 2.24. Input Tape

Word*	Record No.1	Record No.2	Record No.3	Record No.4	Record No.5	Record No.6
0	x	U13(n-1)	0	W2U21(n-2)	W2U21(n-1)	x
1	x	U23(n-1)	0	W2U31(n-2)	W2U31(n-1)	x
2	x	U33(n-1)	0	W2U13(n-2)	W2U13(n-1)	x
3	x	x	0	W2U23(n-2)	W2U23(n-1)	W3
4	x	x	0	W2U33(n-2)	W2U33(n-1)	x
5	x	x	0	W3U12(n-2)	W3U12(n-1)	x
6	x	U11(n-1)	0	W3U22(n-2)	W3U22(n-1)	x
7	x	U21(n-1)	0	W3U22(n-2)	W3U32(n-1)	x
8	x	U31(n-1)	0	W3U11(n-2)	W3U11(n-1)	x
9	x	U13(n-1)	0	W3U21(n-2)	W3U21(n-1)	x
10	x	U23(n-1)	0	W3U31(n-2)	W3U31(n-1)	A3
11	x	U33(n-1)	0	0	0	A2
12	x	U12(n-1)	0	0	0	x
13	x	U33(n-1)	0	0	0	x
14	x	U32(n-1)	0	0	0	x
15	x	U11(n-1)	0	0	0	A1
16	x	U21(n-1)	0	0	0	x
17	x	U31(n-1)	0	0	0	x
18	x	U13(n-1)	0	0	0	x
19	x	U23(n-1)	0	0	0	W1
20	U33(n-1)	x	W1U13(n-2)	W1U13(n-1)	x	x
21	U12(n-1)	Δ U12(n-1)	W1U23(n-2)	W1U23(n-1)	x	1A1
22	U22(n-1)	Δ U22(n-1)	W1U33(n-2)	W1U33(n-1)	x	3A3
23	U32(n-1)	Δ U32(n-1)	W1U12(n-2)	W1U12(n-1)	W2	3A2
24	U11(n-1)	0	W1U22(n-2)	W1U22(n-1)	x	5W3
25	U21(n-1)	0	W1U32(n-2)	W1U32(n-1)	x	x
26	U31(n-1)	0	W2U11(n-2)	W2U11(n-1)	x	8W2

*32 bits each
x - don't care
(n) = result of nth iteration

Figure 2-25, Drum Fill Information

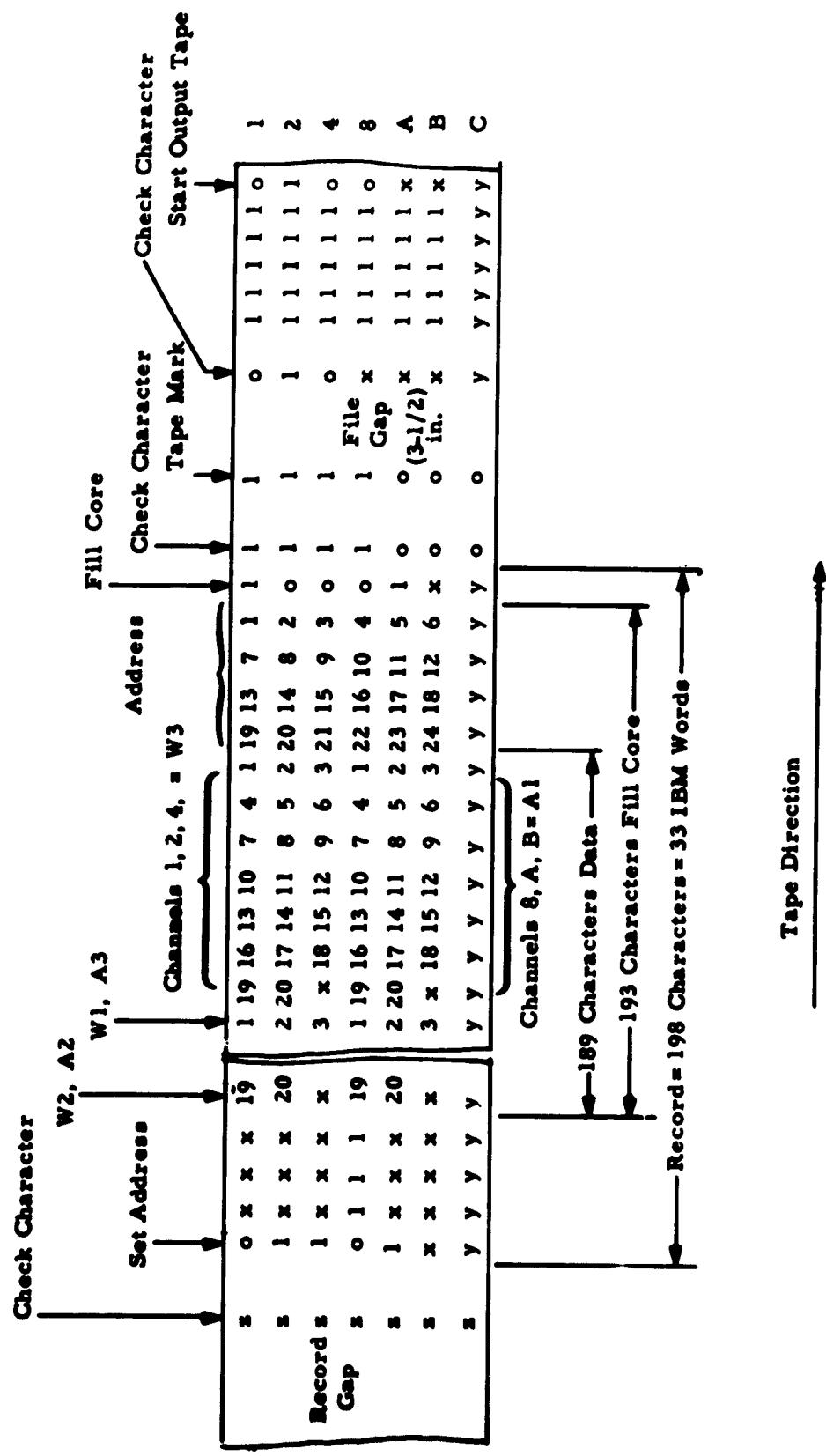


Figure 2-26. Input Tape

Channels

Word	Characters	1, 2, 4	8, A, B
0	1-7	W3	A1
1	8-14	W1*	A3
2	15-21	W2	A2
3	22-28	W3	A1
4	29-35	W1	A3
5	36-42	W2*	A2
6	43-49	W3	A1
7	50-56	W1	A3
8	57-63	W2	A2
9	64-70	W3	A1
10	71-77	W1	A3
11	78-84	W2	A2
12	85-91	W3*	A1
13	92-98	W1	A3
14	99-105	W2	A2
15	106-112	W3	A1
16	113-119	W1	A3
17	120-126	W2	A2
18	127-133	W3	A1
19	134-140	W1	A3*
20	141-147	W2	A2*
21	148-154	W3	A1
22	155-161	W1	A3
23	162-168	W2	A2
24	169-175	W3	A1*
25	176-182	W1	A3
26	183-189	W2	A2

* These values correspond in time.

Figure 2-27 Input Data Organization

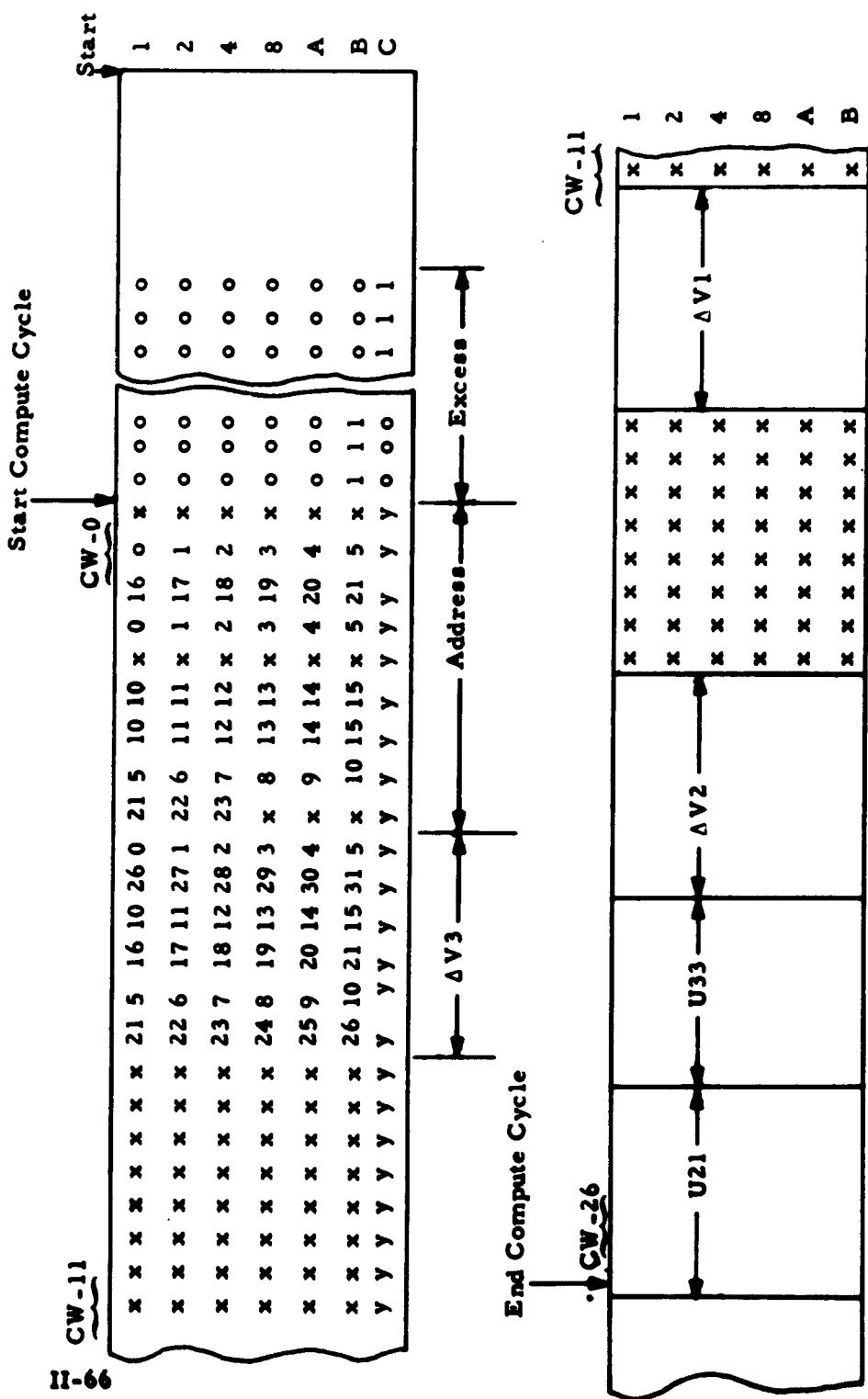


Figure 2-28. Output Tape

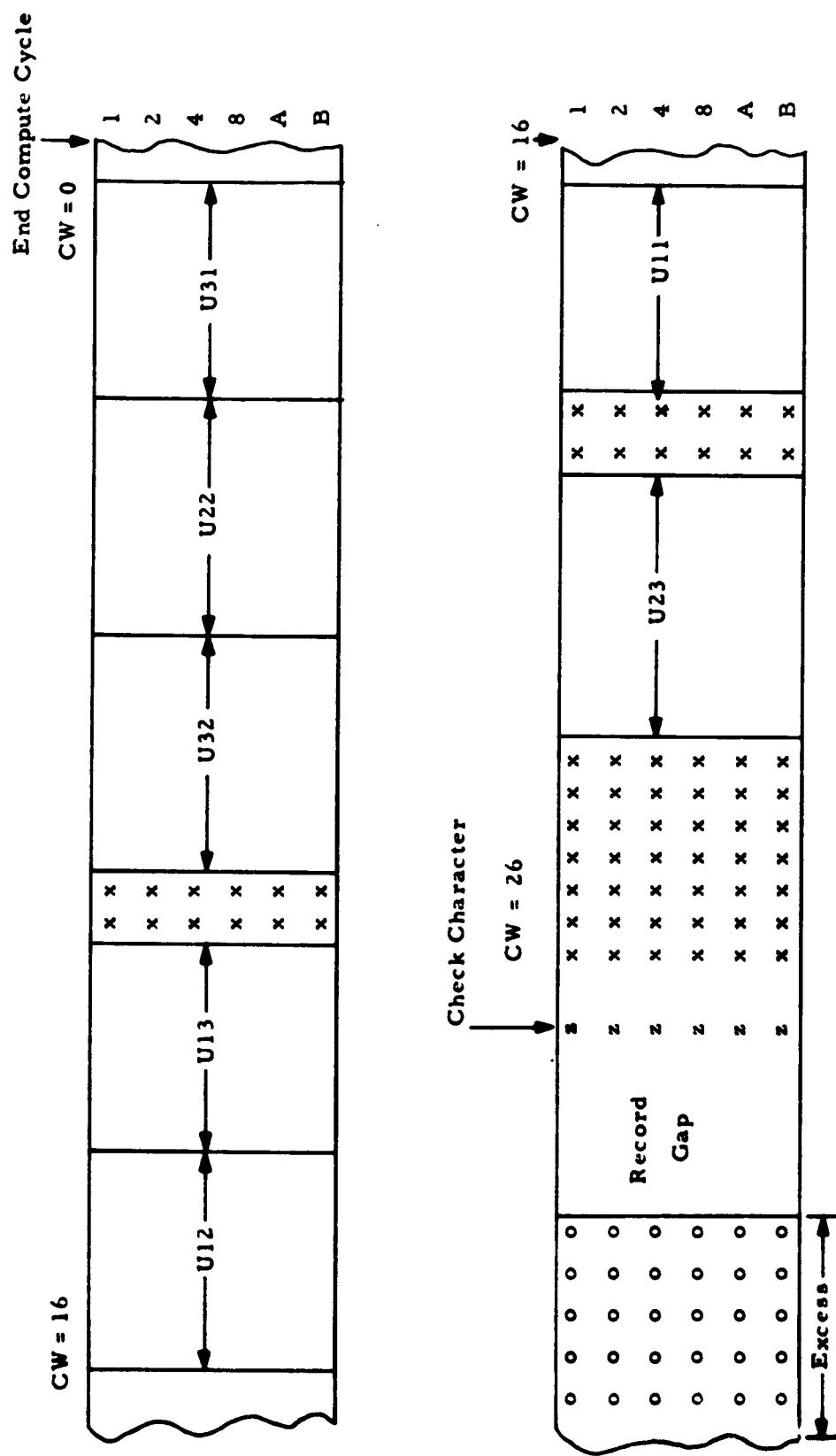


Figure 2-29. Output Tape

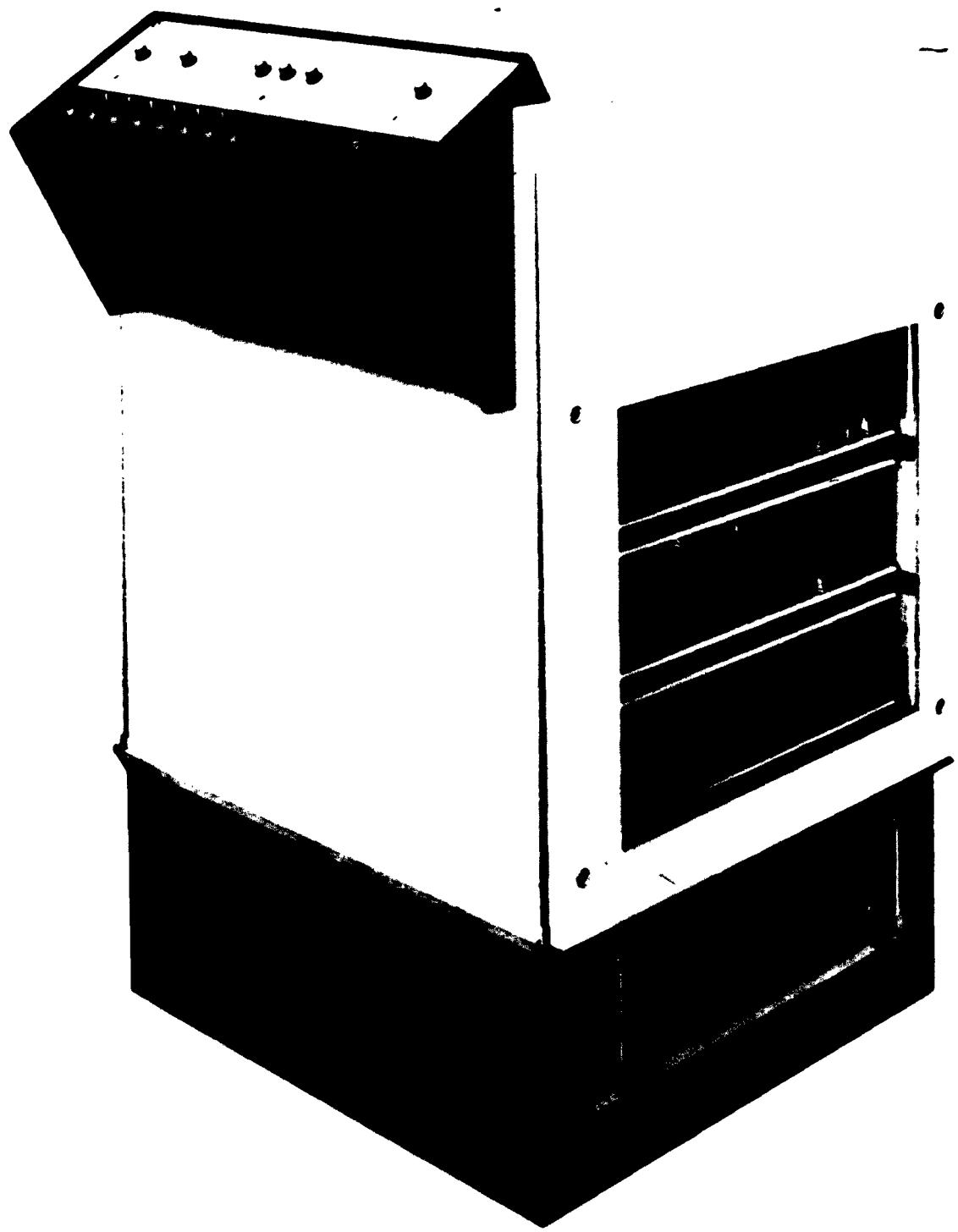


Figure 2-30. Laboratory Model of the HSDDA Computer

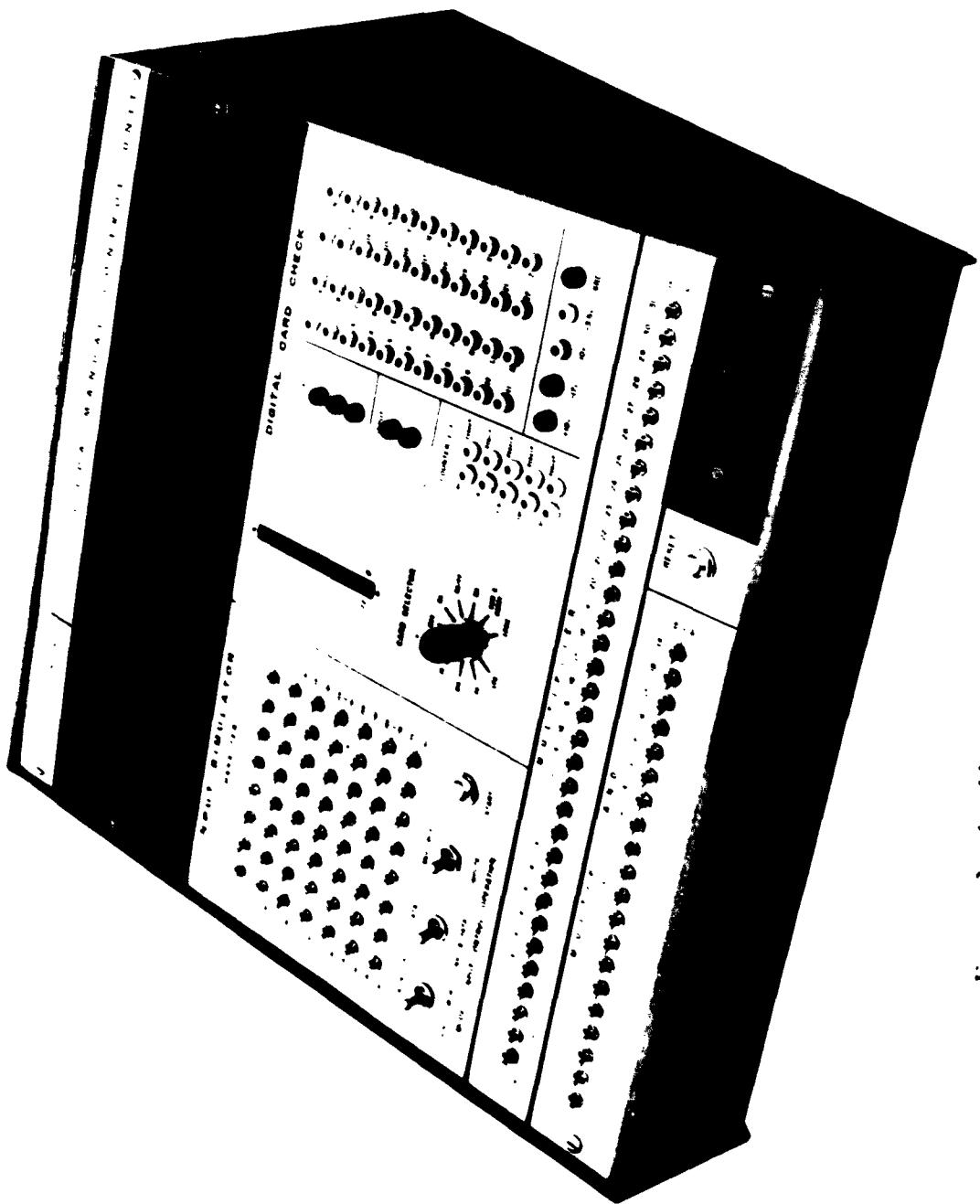


Figure 2-1. Minuteman Control Unit for the MISDDA Computer

2.10 STRAP-DOWN PROCESSOR CHECKOUT AND PERFORMANCE EVALUATION

A. General Features of Checkout and Performance Evaluation

Programs and Tapes - Operation of the breadboard HSDDA was verified to be in accordance with the logical design described. Test tape programs and preparation for evaluation of the HSDDA in the strap-down function was not fully debugged within the program effort period. It is recommended that a brief further effort be provided to evaluate HSDDA performance. For purposes of clarity Figure 2-32 presents the total flow of information processing required for final evaluation and demonstration of the high speed DDA hardware and in block diagram form indicates the requirements for the five 704 programs which are described below.

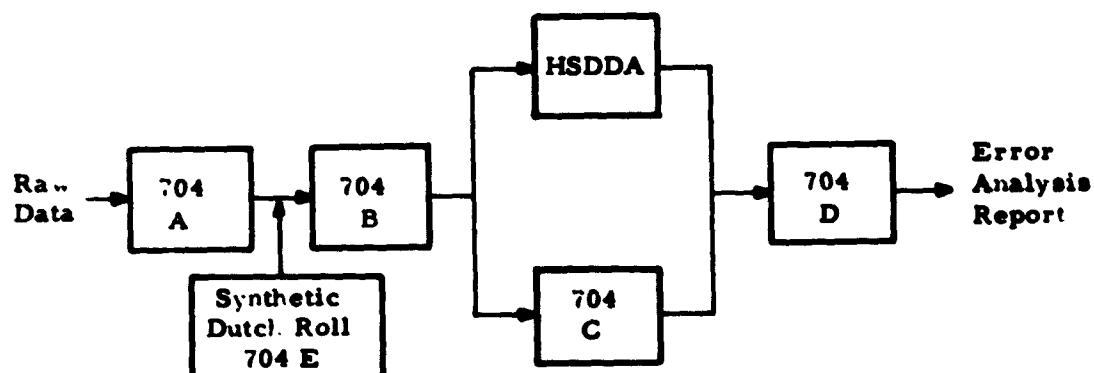


Figure 2-32. Information Processing for Evaluation of High Speed DDA Hardware.

B. IBM 704 Programs

1. The A Program was planned to process raw data provided by WADD into a form compatible with the instrumentation assumptions made in establishing the input characteristics of the high speed DDA. This program accounts for data sample rates, scale factors, resolutions, phase shifts, etc., which may be inherent in the raw data and which must be compensated for before the data can be assimilated by the HSDDA. This program was never written, since no raw data was provided to Litton by WADD.
2. The B Program accepts data in proper analytic form, and provides required phase shifts, formatting, and control signal insertion. The output of this program provides tape which is directly acceptable by the HSDDA.
3. The C Program accepts a tape produced by the B Program and provides a higher accuracy integration process than that used in the HSDDA to produce a set of "yardstick" computations used for performance evaluation of the HSDDA. Care is taken that the phase shifts introduced in the B Program for HSDDA input are properly compensated for in the C Program.
4. The D Program accepts the output tape from the HSDDA and the "yardstick" computations derived as an output tape from the C Program, and provides an error analysis of HSDDA performance.
5. See C. below.

C. Tests Planned For Strap-down Processor Evaluation

1. **Functional Tests-** The functional tests were designed to prove that the hardware was operating in the manner that the logical designer had intended it to operate. Magnetic tapes for these tests required the B Program.
2. **Mechanization Tests -** The mechanization tests were designed to prove that the arithmetic operations built into the HSDDA had indeed provided a valid mechanization of the integration algorithms which the machine was supposed to contain. These tests required the existence of the B Program and also required hand-computed check values which were computed on the basis of the fundamental algorithms of the machine, but without utilizing the machine mechanization.
3. **Algorithm Validity Tests -** These tests were designed to prove the validity of the algorithms which had been mechanized in the HSDDA. They were based on the generation of an input tape which simulates a Dutch roll environment. The Dutch roll is chosen because the closed analytic solution is known, so that arbitrary check point computations are readily obtainable. The first phase of these tests showed that it is possible to demonstrate HSDDA performance through the use of check points hand-computed from the analytic solution. Through the proper choice of amplitude, frequency, and phase of the Dutch roll components it should be possible to make direct visual comparison between two iterations, separated by a specific number of iterations, without having to go to any hand calculations. This effort requires Program E, the

Dutch Roll Generating Program. Programs C and D were completed so that the Dutch Roll Program could be played through the high accuracy integration program and perform error analysis. Then there would be comparative results of the HSDDA, the "yardstick" calculations, and the theoretical analytic solution.

- D. Real Data Tests** - These tests were designed to demonstrate machine performance when operating on real data which was to be provided by WADD. These tests were not executed for the reasons described above.

2.11 FUNCTIONAL AND MECHANIZATION TESTS OF THE STRAP-DOWN

PROCESSOR - The earliest stage of testing of the computer sought to verify the final mechanization operation with regard to the logical designer's intent of basic logical operation and the system analyst's intent of arithmetic processing by the computer. The first of these tests were of an elementary nature consisting of testing the response to streams of alternating 1's and 0's for selected inputs and debugging on this level. In order to confirm the arithmetic processing consistency of the machine for a desired function with a high level of confidence, a set of relatively complicated inputs were selected for which the exact desired outputs could be hand computed by the system analyst over several iterations. To verify the second order algorithm, or any component, requires at least 3 iterations on an input variable of at least the complexity of a quadratic function. To verify communication the quadratic inputs should be distinct for each input variable. The following quadratic family was constructed on this basis:

$$I^{s,w} = (-1)^s \cdot 2^{-w} \left[1 + (-1)^s w^2 - \{7+w-1/2(3-s)(2-s)\} \right]^s$$

where $s = 1, 2, 3$ for coordinates 1, 2, 3

$w = 0, 1$ for acceleration or angular rate input

$w =$ word time of input

Since inputs are sampled at a high rate, and preprocessed by summation of 9 values, the hand computations are simplified by an analytic evaluation of the preprocessing output which is

$$\Delta J_x^{sw} = 9(-1)^s 2^{-w} \left[64 + (-1)^s v^{s,w} e_x^{sw} + (v^{sw})^s \{6 + (e_x^{s,w})^s\} 2^{-s} \right]$$

where $v^{sw} = 2^{-wt+5}$

$$e_x^{sw} = 27x - 15 + a^{s,w}$$

where $a^{s,\omega}$ is the word time at which the input is absorbed in the computer, (deducible from the pertinent table in the logical design description). The input accumulator outputs are then quite simply computed, rounded off, and multiplied by the proper U value of the set specified for tape preparation in the test. The U values used were arbitrary, non-trivial, binary numbers. The products obtained by an octal desk computer were rounded off and put through the desired extrapolator computation to yield the integral increment for each variable, debugging of the computer proceeded on the first iteration. When all desired results were obtained exactly, the next iterations were hand computed, to verify the exact computation of results by the strap-down processor as being for the above inputs.

$$v_n^{jtw} = v_{n-1}^{jtw} + \Delta v_n^{jtw} \quad \text{where } j, t = 1, 2, 3; \omega = 1, 0$$

$$\Delta v_n^{jtw} = \left[\delta_w \delta_r^{\omega} s^{rst} + \delta_t \delta_w \delta_r^{\omega} \right] \left[23P_n^{jrs\omega} - 16P_{n-1}^{jrs\omega} + 5P_{n-2}^{jrs\omega} \right] z^{-7}$$

$$P_n^{jrs\omega} = R \left[v_{n-1}^{jri} \cdot \phi_n^{s,\omega} \right], R = \text{precision round-off to 19 bits \& sq}$$

$$\phi_n^{s,\omega} = 9 \Delta J_n^{s,\omega} - \Delta J_{n-1}^{s,\omega}$$

$$\Delta J_n^{s,\omega} = \sum_{k=0}^6 I_k^{s,\omega}(w) \quad \text{where } w = 27(n-1) + 3k + \eta^{s,\omega}$$

A. Algorithm Validity Tests.

1. Introduction - The accuracy of the strap-down processor for a given set of input functions is determined, in practice, by comparison of processor outputs with other computed values which insofar as possible are nearly exact values or at least of significantly higher accuracy than can be expected of the processor. Two approaches enable computation of "exact" or "yardstick" solutions for certain output types and have valuable coordinated uses. In the case of simple Dutch roll motion generated by inputs which are expressable as analytic functions of the type presented in Chapter 6 Section 4 of the

Phase I report, the exact inertial reference is expressed as an explicit function so that evaluation of the processor can be as exact as desired at any point, by hand or machine computation, without using incremental computation techniques subject to error build-up. This approach is the most incontrovertible for the limited class of inputs for which it is feasible. The second approach is the general solution by incremental computation of very high accuracy which in principle can be attained because numerical integration with any n^{th} order algorithm ($n \geq 0$) has accuracy which generally increases with decreased iteration interval. The second approach however, is subject to accuracy limitations because the actual computation is not performed with whole numbers of unlimited accuracy as required by theoretical considerations, but with a whole number of limited bit length. There are other practical problems such as increase in costs resulting from the decreased iteration interval. The next section of this report presents the ECD programs which enable evaluation of the processor for Dutch roll and provide a general "yardstick" incremental computation. The validity of the latter should be checked by comparison of the exact analytic solution for Dutch roll. Provided the comparison is found sufficiently close compared to processor output, it can be assumed that round-off and algorithm error in the C Program is not a problem for low frequency variables. The sensitivity to noise of the yardstick computation can probably be evaluated analytically on the one hand and by yardstick program runs with contracted or expanded iteration interval on the other. If it is found that two widely different but small iteration intervals yield substantially the same results the really general accuracy of the proposed yardstick computation program is assured with respect to random rather than bias errors. These latter analyses were not performed during this program.

2.12 THE DUTCH ROLL INPUT, GENERAL YARDSTICK COMPUTATION, AND PROCESSOR-ERROR EVALUATION (ECD) PROGRAMS.

A. General Description - The following is a description and listing of the IBM 704 ECD Program for the High Speed DDA. The ECD Program simulates the HSDDA and compares the simulation with the true output of the HSDDA.

The E section of the ECD Program generates six inputs.

Case I - $\omega_1 = A \sin (\theta_{03} \omega t n_3 \omega t + \theta)$ (II-38)

$$\omega_2 = A \cos (\theta_{03} \omega t n_3 \omega t + \theta)$$

$$\omega_3 = 0$$

$$A_1 = 0$$

$$A_2 = 0$$

$$A_3 = 0$$

Case II - $\omega_1 = 0$ (II-39)

$$\omega_2 = A \sin (\theta_{03} \omega t n_3 \omega t + \theta)$$

$$\omega_3 = A \cos (\theta_{03} \omega t n_3 \omega t + \theta)$$

$$A_1 = 0$$

$$A_2 = 0$$

$$A_3 = 0$$

Case III - $\omega_1 = A \cos (\theta_{03} \omega t n_3 \omega t + \theta)$ (II-40)

$$\omega_2 = 0$$

$$\omega_3 = A \sin (\theta_{03} \omega t n_3 \omega t + \theta)$$

Case III - $A_1 = 0$
 (cont)
 $A_2 = 0$
 $A_3 = 0$

The values for the constants and variables are:

$$\theta_{03}^{\omega t} = 0.0004882812694$$

$$A = -0.6666663342844$$

$$B = -0.002502441124$$

$n_3 \omega t$ = A positive integer starting from zero and increasing by one (1) every iteration of the 704 program.

It will be noticed that a sine and cosine polynomial has to be generated every iteration. Since thousands of iterations will be performed, the above equations will be generated once every 50 iterations. During the intervening iterations, the following equations are used for the sine and cosine:

$$S_n = S_{n-1} \cos \theta_{03}^{\omega t} + C_{n-1} \sin \theta_{03}^{\omega t} \quad (\text{II-41})$$

$$C_n = C_{n-1} \cos \theta_{03}^{\omega t} - S_{n-1} \sin \theta_{03}^{\omega t} \quad (\text{II-42})$$

Where S_n and C_n are the sine and cosine for the present iteration, S_{n-1} and C_{n-1} are the sine and cosine for the previous iteration.

$$\cos \theta_{03}^{\omega t} = 0.9999998809$$

$$\sin \theta_{03}^{\omega t} = 2^{-11}$$

Since there is a possibility that the argument for the sine and cosine can become increasingly large and cause overflow due to the variable $n_3 \omega t$, the argument is tested against $\pi/2$. When it exceeds $\pi/2$, the excess becomes the new argument. Consequently, there will also be a change in the signs of the sine and cosine.

The C section of the ECD Program receives its inputs from the E Section and evaluates the following set of equations which are also solved by the High Speed DDA.

$$\Delta V^*_{j,1} = \alpha_1 U_{j,1} + \alpha_2 U_{j,2} + \alpha_3 U_{j,3} \quad j = 1..3 \quad (\text{II-43})$$

$$\Delta V_j = \frac{23}{12} \Delta V^*_{j,n} - \frac{16}{12} \Delta V^*_{j,n-1} + \frac{5}{12} \Delta V^*_{j,n-2} \quad j = 1..3 \quad (\text{II-44})$$

$$\Delta U^*_{j,1} = \omega_3 U_{j,3} - \omega_2 U_{j,2} \quad j = 1..3 \quad (\text{II-45})$$

$$\Delta U^*_{j,2} = \omega_1 U_{j,3} - \omega_3 U_{j,1} \quad j = 1..3 \quad (\text{II-46})$$

$$\Delta U^*_{j,3} = \omega_2 U_{j,1} - \omega_1 U_{j,2} \quad j = 1..3 \quad (\text{II-47})$$

$$\Delta U^*_{j,1} = \frac{23}{12} \Delta U^*_{j,1} - \frac{16}{12} \Delta U^*_{j,1,n-1} + \frac{5}{12} \Delta U^*_{j,1,n-2} \quad j = 1..3 \quad (\text{II-48})$$

$$U_{j,2} = \frac{23}{12} \Delta U^*_{j,2} - \frac{16}{12} \Delta U^*_{j,2,n-1} + \frac{5}{12} \Delta U^*_{j,2,n-2} \quad j = 1..3 \quad (\text{II-49})$$

$$U_{j,3} = \frac{23}{12} \Delta U^*_{j,3} - \frac{16}{12} \Delta U^*_{j,3,n-1} + \frac{5}{12} \Delta U^*_{j,3,n-2} \quad j = 1..3 \quad (\text{II-50})$$

$$\begin{aligned} *U_{Nj,1} &= U_{j,1} + \Delta U_{j,1} \\ *U_{Nj,2} &= U_{j,2} + \Delta U_{j,2} \quad j = 1..3 \\ *U_{Nj,3} &= U_{j,3} + \Delta U_{j,3} \end{aligned} \quad (\text{II-51})$$

* U_N = new value of U_j

B. Program Execution - The $U_{j,k}$ array is initialized before any computations are made. Then control numbers for a tape input α_k , mode or case selection (1, 2, 3) for input of ω_K are zero. The case selection refers to cases I, II, III for selection of the order of sines and cosines (i.e., ω_1 , ω_2 , ω_3) described previously. The calculations are performed in double precision floating point except for the sines and cosines generated by the Dutch roll routine. The fixed point values of the Dutch roll (ω_1 , ω_2 , ω_3) are floated and used in the double precision calculations. The output quantities are fixed and sent to an output area where they are printed in the D Program.

The D section of the ECD Program prints and/or compares the output from the High Speed DDA and the output from the C section of the program. The D section of the program for the High Speed DDA has three modes of operation:

1. Comparison of the 704 tape (output of the C Program) and the High Speed DDA tape (output of the High Speed DDA) and listing of the corresponding elements of each matrix, and the difference between the corresponding elements in decimals.
2. Listing of the elements of the High Speed DDA tape in both octal and decimal.
3. Listing of the elements of the 704 tape in decimal.

The interval between printing and the number of consecutive records that are printed can be selected. A general flow chart of the D section of the program is presented in Figure 2-33.

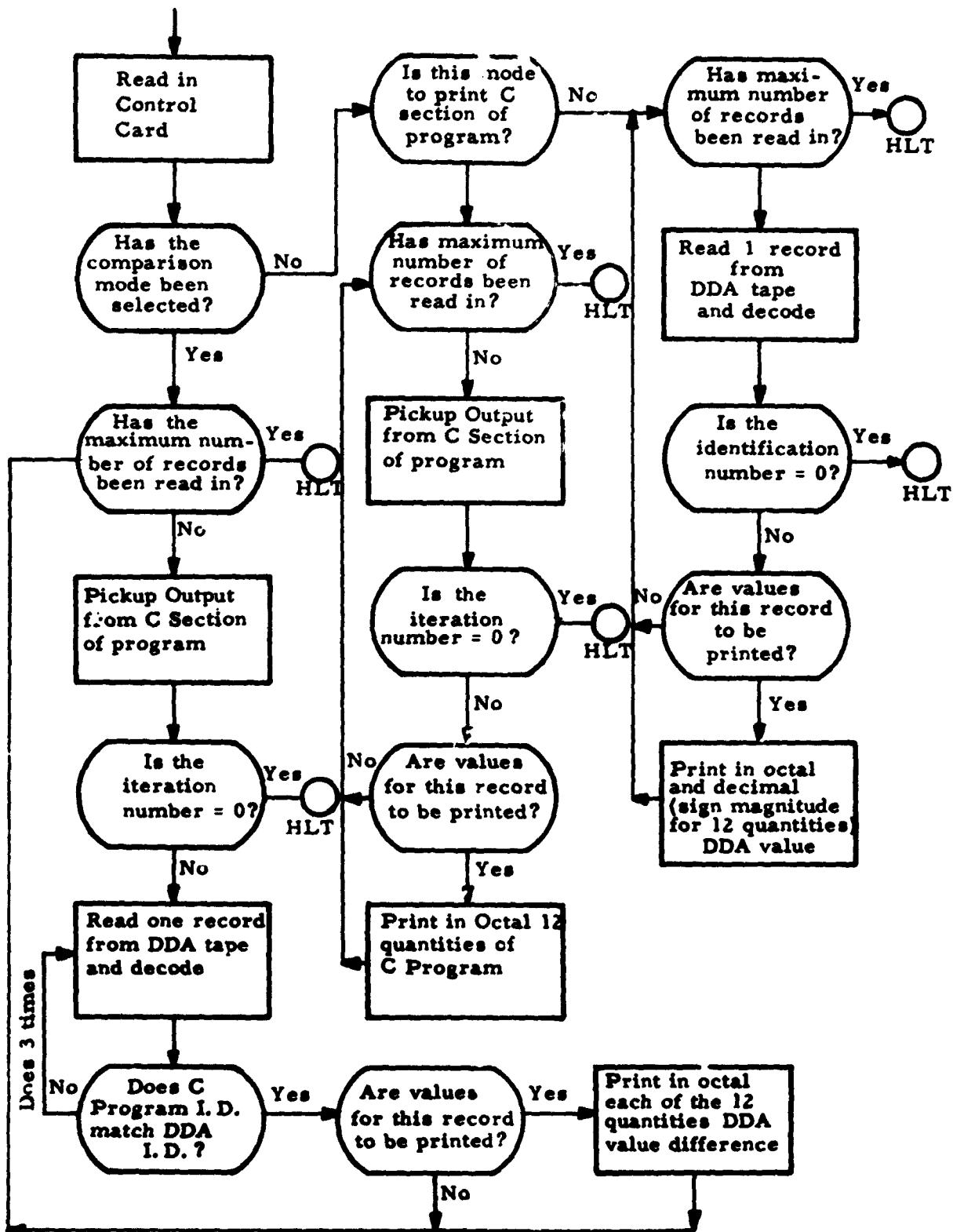


Figure 2-33. General Flow Chart

C. Control Cards - Placed After Transfer Card.

Card 1:

Column 1	- Mode
1	- Compares DDA tape and DDA simulation program and prints difference
2	- Simulates DDA and prints result
3†	- Prints DDA tape (on Unit no. 4)

**Starts in Column 2 - Skip interval (in decimal) must be followed
by a comma**

**Next Column - Print interval (in decimal) must be followed
by a comma**

**Next Column - Maximum number of records (in decimal); to
process next column must be blank**

**Example: Column - 1 2 3 4 5 6 7 8 9 10 11 12
Punch - 1 5 8 , 9 , 8 0 4**

**Mode = 1, (which compares DDA tape and DDA simulation and
prints the difference)**

**Skip Interval = 58, (after every 58 iterations, 9 consecutive itera-
tions will be printed)**

Print Interval = 9, (9 consecutive iterations will be printed)

**Maximum Number of Records = 804, (after processing 804 records
"iterations" the program will halt)**

†When using Mode 3, no more control cards are necessary.

Card 2:

Column 1 - Tape Number (A-Program input, not used yet)

Column 2 - Case Number for Dutch roll, a 1, 2, or 3 must be punched in this column

Column 3 - Must be a comma

Starts in Column 4 - Identification Number[†] (in decimal) column following must be a comma

Next Column - Iteration Number[‡] (in decimal) column following must be blank

Example: Column - 1 2 3 4 5 6 7 8 9 10 11 12

Punch - 1 1 , 3 , 1 8 7 2 4

Tape Number = 1 (not used) can be anything

Case Number = 1

Identification Number = 3

Iteration Number = 18724₁₀ (DDA iteration number must be equal to 44444₈ in octal)

Control Cards 3-35: The following 33 cards are for initialization and each card must be included even if the value is zero.

Column 1 - Blank

[†]When mode number 1 is used (control card 1), the identification of the DDA tape (1st record) and the identification punched in control card 2 must be identical.

[‡] When mode number 1 is used (control card 1), the iteration number on the DDA tape (record 2) and the iteration number punched in control card 2 must be identical.

Column 2 - If the sign of the decimal value is negative, then a (-) sign must be placed in this column and a decimal point (.) is placed in Column 3

If the sign of the decimal value is positive, the decimal point (.) is punched in this column

Column 3 or 4 - The decimal value starts in this column. The decimal exponent follows immediately after the character E (which follows immediately after the last digit of the decimal value). If the character E does not appear, the exponent is assumed to be zero.

Example: Column - 1 2 3 4 5 6 7 8 9 10

- . 0 3 4 6 2

. 3 4 6 2 E - 1

(. 3462E-1 is the same as . 03462)

Card 3 - ΔV^* 1 n-1

Card 4 - ΔV^* 2 n-1

Card 5 - ΔV^* 3 n-1

Card 6 - ΔV^* 1 n-2

Card 7 - ΔV^* 2 n-2

Card 8 - ΔV^* 3 n-2

Card 9 - ΔU^* 11 n-1

Card 10 - ΔU^* 12 n-1

Card 11 - ΔU^* 13 n-1

Card 12 - ΔU^* 21 n-1

Card 13 - ΔU^* 22 n-1

Card 14 - ΔU^* 23 n-1

Card 15 - ΔU^* 31 n-1

Card 16 - ΔU^* 32 n-1

Card 17 - ΔU^* 33 n-1

Card 18 - ΔU^* 11 n-2

Card 19 - ΔU^* 12 n-2

Card 20 - ΔU^* 13 n-2

Card 21 - ΔU^* 21 n-2

Card 22 - ΔU^* 22 n-2

Card 23 - ΔU^* 23 n-2

Card 24 - ΔU^* 31 n-2

Card 25 - ΔU^* 32 n-2

Card 26 - ΔU^* 33 n-2

Card 27 - U 11	Card 32 - U 23
Card 28 - U 12	Card 33 - U 31
Card 29 - U 13	Card 34 - U 32
Card 30 - U 21	Card 35 - U 33
Card 31 - U 22	

D. Operating Procedures. (Tapes)

<u>Unit</u>	<u>Purpose</u>
4	High Speed DDA tape
6	Off line output if sense switch 2 is OFF (all others not used)
<u>Sense Switches</u>	
1	ON
2	ON for on line printing OFF for off line printing to go on Tape Unit no. 6
3	OFF
4	OFF
5	OFF
6	OFF

The SHARE 2 Printer board is used. The Program does not rewind any tapes.

E. Programmed Halts.

- 33143 - Control cards missing (Card no. 1)
- 33144 - Control cards incorrect (Bad punch) (Card no. 1)
- 34032 - Control cards missing (Card no. 2)
- 34033 - Control cards incorrect (Bad punch) (Card no. 2)

34051 - Control cards missing (Cards 3-8)
 34052 - Control cards incorrect (Bad punch) (Cards 3-8)
 34062 - Control cards missing (Cards 9-26)
 34063 - Control cards incorrect (Bad punch) (Cards 9-26)
 34073 - Control cards missing (Cards 27-35)
 34074 - Control cards incorrect (Bad punch) (Cards 27-35)
 34564 - End of File Mark on HSDDA tape found while reading data (Mode 1)
 34605 - Iteration numbers do not match - tried 3 times
 34636 - Iteration number = 0 (Mode 1)
 34665 - Iteration number = 0 (Mode 2)
 34706 - End of File Mark on HSDDA tape found while reading data
 34742 - Iteration number = 0 (Mode 3) to restart transfer to 34671
 35044 - Maximum number of records have been processed as specified in control card 1
 35603 - Identification of HS DDA tape and identification number punched on Control Card 2 are not identical
 35602 - No identification number has been punched on Control Card 2

2.13 INITIALIZATION OF THE STRAP-DOWN PROCESSOR - The ΔU_{jk}^* generated by the HSDDA are approximations to the central differences indicated in Figure 2-34.

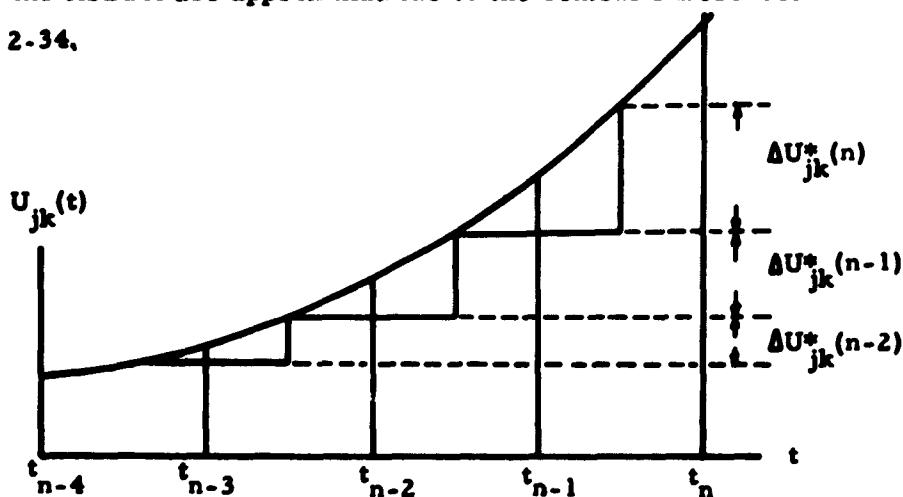


Figure 2-34 Central Differences for ΔU_{jk}^*

The extrapolation formula

$$\Delta U_{jk} \approx \frac{23}{12} \Delta U_{jk}^*(n) - \frac{16}{12} \Delta U_{jk}^*(n-1) + \frac{5}{12} \Delta U_{jk}^*(n-2) \quad (\text{II-52})$$

is used to approximate

$$\Delta U_{jk} = U_{jk}(t_n) - U_{jk}(t_{n-1}). \quad (\text{II-53})$$

Initialization requires the computation of the ΔU_{jk}^* 's or, equivalently, of products (involving angular rates) which sum to them. An alternate approach requiring values of the U_{jk} 's only, is to compute suitable ΔU_{jk}^* 's from the formulas

$$\begin{aligned} \Delta U_{jk}^*(1) &= \frac{1}{2} U_{jk}(t_1) - U_{jk}(t_{-1}) \\ \Delta U_{jk}^*(0) &= \Delta U_{jk}^*(1) - U_{jk}(t_1) - 2U_{jk}(t_0) + U_{jk}(t_{-1}) \\ \Delta U_{jk}^*(-1) &= \Delta U_{jk}^*(0) - U_{jk}(t_1) - 2U_{jk}(t_0) + U_{jk}(t_{-1}) \end{aligned} \quad (\text{II-54})$$

These formulas are exact if $U_{jk}(t)$ is a quadratic. Furthermore, elimination of the ΔU_{jk}^* 's between (II-54) and (II-52) does indeed yield (II-53) exactly.

During the first iteration the computer computes ΔU_{jk}^* (II-54) from initially given U_{jk} 's and ω 's. It is possible, however, to determine ω 's from the $U_{jk}(t_0)$'s which will generate the desired values as given by (II-54). The necessary equations are

$$\begin{aligned} U^3 \omega_1 &= U_{j3}(t_0) \Delta U_{j2}^*(1) - U_{j2}(t_0) \Delta U_{j3}^*(1) \\ U^3 \omega_2 &= U_{j1}(t_0) \Delta U_{j3}^*(1) - U_{j3}(t_0) \Delta U_{j1}^*(1) \\ U^3 \omega_3 &= U_{j2}(t_0) \Delta U_{j1}^*(1) - U_{j1}(t_0) \Delta U_{j2}^*(1) \end{aligned} \quad (\text{II-55})$$

$$\text{where } U^2 = U_{j1}^2 + U_{j2}^2 + U_{j3}^2.$$

If computations (in the HSDDA) are to be performed for more than one inertial axis, (II-55) is not sufficient, since it determines a set of ω 's for which the rotation about the j th inertial axis is zero. What is necessary for an orthogonal inertial system is to calculate an ω -vector from (II-55) for each j . The appropriate ω -vector to be used for the first iteration is then given by half the sum of the three vectors so calculated. The multiplicative factor $8/9$ must also be included in the scaling constants to defeat the extrapolation in the input accumulator unit.

2.14 PROGRAMMING METHODS FOR PROCESSOR EVALUATION FOR REAL DATA INPUTS - Real data which is recorded during actual flight is not expected to be in a form directly assimilatable by the strap-down processor. In addition to recording format, there are more fundamental differences stemming from digital representation and sensor-transducer accuracy. The latter problem is overcome by making yardstick calculations on the same transformed real data that is input to the strap-down processor. The problem of transforming data recorded by, say, a pulse stream analogue to digital converter to the form in which a whole word sampler would present the real data is analyzed in the next section.

2.15 INTERPOLATION CALCULATIONS ON WADD SUPPLIED DATA FOR GENERATION OF STRAP-DOWN PROCESSOR INPUT DATA.

A. **Proposed Methods** - Data supplied by WADD for strap-down processor evaluation which is the recorded output of a pulse stream analogue to digital converter must be put in a form assimilatable by the breadboard processor (it must be subjected to short lag smoothings corresponding to analogue filtering in the required analogue setup of the strap-down system assumed in strap-down processor design).

Table I. Initialising Constants for Dutch Roll

Octal	Value	Decimal	Location (Record & Word)		
			Case 1	Case 2	Case 3
3000000 0000000	0.75	2-0 U _{1,3}	1-25 U _{3,1}		1-23 U _{3,2}
3000000 0000000	0.75	2-9 U _{1,3}	2-7 U _{3,1}		
3000000 0000000	0.75	2-18 U _{1,3}	2-16 U _{3,1}		2-14 U _{3,2}
0000000 7460000	0.000007 241964	3-20 ω ₁ U _{1,3}	4-0 ω ₃ U ₃		4-7 ω ₃ U _{3,2}
0000000 1714000	0.000001 810499	3-26 ω ₂ U _{1,1}	4-6 ω ₃ U _{3,1}		3-22 ω ₁ U _{3,2}
000153 7776400	0.000823 971971	4-5 ω ₃ U _{1,3}	3-21 ω ₁ U _{3,2}		4-1 ω ₂ U _{3,1}
0000000 3630000	0.000003 620982	4-20 ω ₁ U _{1,3}	5-0 ω ₃ U _{3,1}		5-7 ω ₃ U _{3,2}
0000000 0746000	0.000000 905246	4-26 ω ₂ U _{1,1}	5-6 ω ₃ U ₃		4-22 ω ₁ U _{3,2}
000153 7776400	0.000823 971971	5-5 ω ₃ U _{1,3}	4-21 ω ₁ U _{3,2}		4-1 ω ₂ U _{3,1}
777500 1226400	-0.001463 609748	5-23 Old ω ₃	6-3 Old ω ₃		6-19 Old ω ₁
777772 2340000	-0.000043 451786	6-3 Old ω ₃	6-19 Old ω ₁		5-23 Old ω ₂
777777 2740000	-0.000004 827976	6-24 5ω ₃			6-26 8ω ₂
777500 011400	-0.001464 702189	6-20 8ω ₃	6-24 5ω ₃		
002777 775240	+0.011718 709953			Multiplicand (-ω ₁)	

All unspecified locations are filled with zero. Octal numbers are in standard 704 format (except that negatives are complemented), and constitute the optimum rounded form of the corresponding decimals. Notice that for Case 1, 9(S_{ω₃}) - (Old ω₃) = 0 and 9(S_{ω₂}) - (Old ω₂) = -0.011718 709953.

Corresponding constants for R. Fine's E Program are:

$$\begin{aligned} A &= -0.666666 291240 \quad B0 = (652\ 525\ 221\ 362)_8 \\ B &= 0.001892 089378 \quad B4 = (000\ 017\ 377\ 777)_8 \end{aligned}$$

The WADD supplied data must be processed by an interpolation calculation which is capable of yielding each appropriate equivalent (or interpolated) data point for the specific sampling time, at which the processor would normally sample real time data represented by the WADD supplied data. Two possible interpolation calculations are derived in the following sections for methods differing in the degree of polynomial assumed and the number of fit data, for each computed data value. The first analysis assumes computations based on step change in slope as a function of system parameters which occur in the three point quadratic fit method. The second analysis evaluates fractional error from that of equivalent inputs for preprocessor section outputs. An error of fractional amplitude $\pm 0.3/24$ (90° out of phase with the input is 0.6×10^{-6} for 1 cps inputs to a processor with 266 it/sec). While the step slope changes at tape data points for a three point quadratic interpolation routine are anappreciable fraction (≈ 1 percent over 1 cps signals) of the total slope, the effect on short integrals is small (0.6×10^{-6} for a 1 cps signal), being at the border line of effecting the 20 bit input to the processor. [#] If the errors were similar to the roundoff error of independent distribution the system error would be small. If the errors were similar to the roundoff error of a bias nature, the system error would be large in one hour. A heuristic analysis based on the assumption that the errors have a correlation time of 0.4 sec for 1 cps inputs leads to a net error of $0.6 \times 10^{-6} \times \sqrt{0.4 \text{ sec}} 360^\circ \text{ sec} = 2.16 \times 10^{-6}$ radians ± 5 arc sec in inertial reference computation after one hour. This is a small but appreciable error. The conjecture that the use of interpolated values for both the processor and for the evaluation of GP solutions removes most

$\pm 0_0 = 2\pi f\tau$, f - frequency, τ - iteration interval of processor outputs.

[#]Assuming maximum angular rate frequency of 1.5 cps.

of the difference of solutions, is complicated if a third order algorithm is used in the latter, since discontinuities occur in the higher differences utilized. These considerations together with the unacceptable complexity of a four point cubic interpolation imply the preferable use of the three point quadratic formula type, whose calculation function is presented in the following section.

B. Derivation of Three Point Quadratic Interpolation Method - A quadratic interpolation formula of the form

$$I_p^* = I_n + \lambda(p-n) + u(p-n)^2 \quad (\text{II-56})$$

is correct at $p = n$. At $p = n+1$, correctness requires

$$I_{n+1} = I_n + \lambda + u \quad (\text{II-57})$$

$$I_{n-1} = I_n - \lambda + u \quad (\text{II-58})$$

for which $u = \frac{\Delta^2 I_{n+1}}{2}$ (II-59)

$$\lambda = \frac{\Delta I_{n+1} + \Delta I_n}{2} \quad (\text{II-60})$$

The quadratic interpolation formula is

$$I_p^* = I_n + \frac{(\Delta I_{n+1} + \Delta I_n)}{2\tau} (p - n) + \frac{\Delta^2 I_n}{2} (p - n)^2 \quad (\text{II-61})$$

which for $p = t/\tau$ has the form

$$I_p^*(t) = I_n + \frac{(\Delta I_{n+1} + \Delta I_n)}{2\tau} (t - n\tau) + \frac{\Delta^2 I_n}{2\tau^2} (t - n\tau)^2 \quad (\text{II-62})$$

Computation of $I^*(t)$ each interval τ/M starting at $t = n\tau$ is investigated in terms of a difference operator $\Delta^*(\cdot)$ defined by $\Delta^* X_p = X(\frac{p\tau}{M}) - X\left[\frac{(p-1)\tau}{M}\right]$. (II-63)

In terms of counts at τ/M intervals where $t = \frac{r\tau}{M}$,

$$I_{r/M}^* = I_n + \frac{(\Delta I_{n+1} + \Delta I_n)}{2M} (r - nM) + \frac{\Delta^2 I_{n+1}}{2M^2} (r - nM)^2 \quad (II-64)$$

The application of the $\Delta^*(\cdot)$ operator yields

$$\Delta^*(I_{r/M}^*) = \frac{(\Delta I_{n+1} + \Delta I_n)}{2M} + \frac{\Delta^2 I_{n+1}}{M^2} [(r - nM) - 1/2] \quad (II-65)$$

$$\Delta^{**}(I_{r/M}^*) = \frac{\Delta^2 I_{n+1}}{M^2}$$

For interpolation from $t = n\tau$ to $t = (n+1)\tau$ at intervals of τ/M , the three numbers, I_n and

$$\alpha_n = \frac{\Delta^2 I_{n+1}}{M^2} \quad (II-66)$$

$$\beta_n = \frac{(\Delta I_{n+1} + \Delta I_n)}{2M} - \frac{\alpha_n}{2} \quad (II-67)$$

are adequate for a sequential generation by pure summations in an interpolator program used in input tape preparation for the strap-down processor. Tape preparation by a computer with slower multiplication operation than two addition times should be less costly using the above calculation procedure than methods using multiplication.

C. Derivation of Four Point Cubic Interpolation - A cubic interpolation formula of the form

$$I_p^* = I_n + \lambda(p - n) + \mu(p - n)^2 + \nu(p - n)^3 \quad (\text{II-68})$$

is correct at $p = n$. At $p = n + 2, N + 1, N - 1$ correctness requires

$$I_{n+2} = I_n + 2\lambda + 4\mu + 8\nu \quad (\text{II-69})$$

$$I_{n+1} = I_n + \lambda + \mu + \nu \quad (\text{II-70})$$

$$I_{n-1} = I_n - \lambda + \mu - \nu \quad (\text{II-71})$$

Adding the last two equations we obtain

$$\mu = \frac{\Delta^3 I_{n+1}}{2} . \quad (\text{II-72})$$

Eliminating λ and substituting μ

$$\nu = \frac{\Delta^3 I_{n+2}}{6} \quad (\text{II-73})$$

also

$$\lambda = \Delta I_{n+1} - \mu - \nu \quad (\text{II-74})$$

The four point cubic interpolation formula has for $p = t/\tau$ the form

$$I^*(t) = I_n + \frac{\lambda}{\tau}(t - n\tau) + \frac{\mu}{\tau^2}(t - n\tau)^2 + \frac{\nu}{\tau^3}(t - n\tau)^3 \quad (\text{II-75})$$

Computation of $I^*(t)$ each interval τ/M , starting at $t = n\tau$, is investigated in terms of a difference operator $\Delta^*(\)$ defined by

$$\Delta^*(x_p) = x(p\tau/M) - x\left[(p-1)\tau/M\right] . \quad (\text{II-76})$$

Substituting $t = \frac{r\tau}{M}$ in the $I^*(t)$ formula

$$I_{r/M}^* = I_n + \frac{\lambda}{M}(r - nM) + \frac{\mu}{M^2}(r - nM)^2 + \frac{\nu}{M^3}(r - nM)^3 \quad (\text{II-77})$$

since

$$\Delta^* (r - nM)^2 = 2[(r - nM) - 1/2] \quad (\text{II-78})$$

$$\Delta^* (r - nM)^3 = 3[(r - nM)^2 - (r - nM) + 1/3].$$

Application of the operator $\Delta^*(\)$ to $I^* r/M$ yields

$$\begin{aligned} \Delta^* I_{r/M}^* &= \frac{\lambda}{M} + \frac{2\mu}{M^2} [(r - nM) - 1/2] + \frac{3\nu}{M^3} [(r - nM)^2 - (r - nM) + 1/3] \\ \Delta^{**} I_{r/M}^* &= \frac{2\mu}{M^2} + \frac{3\nu}{M^3} [2(r - nM) - 2] \\ \Delta^{***} I_{r/M}^* &= + \frac{6\nu}{M^3} \end{aligned} \quad (\text{II-79})$$

$$\text{denoting} \quad a_n = 6\nu n / M^3$$

$$a_n = \frac{2\mu n}{M^2} - \frac{6\nu n}{M^3}$$

where λ, μ, ν (for interval $t = n\tau$ to $(n+1)\tau$) are computed in the manner derived. Then

$$\Delta^{**} I_{r/M}^* = a_n + \sum_{r^* = nM+1}^r a_{r^*} \quad (\text{II-80})$$

$$\Delta^* I_{r/M}^* = a_n + \sum_{r^* = nM+1}^r \Delta^{**} I_{r^*/M}^* \quad (\text{II-81})$$

$$I^*_r/M = I_n + \sum_{r^* = nM+1}^r \Delta * I^*_{r^*}/M \quad (\text{II-82})$$

define a program computation requiring three add times per iteration.

CHAPTER III

ANALYTICAL DEVELOPMENTS DURING PHASE II IN THE GENERAL THEORY OF REAL TIME COMPUTATION

3.0 MOTIVATION OF INVESTIGATIONS AND APPLICATIONS OF RESULTS-

The relations of input variables of the external world to the internal operation of a computer, subject to certain general constraints in numerical operation, provide a basis for development of a general theory of real time computation which may otherwise have an abstract nature in that mechanization is to be deduced subsequently (independently taking into account hardware factors) in applying the principles developed. The feasibility and use of such general investigations stems from the fact that a computer is designed to accomplish the analytical task. A major analytical development during Phase I was the derivation of a theory of numerical integration applying to the most general type of integration of an incremental computer (actually appropriate in nearly all applications) which is of the classical operation of Stieltjes' integration. The algorithms developed for precision integration of the Stieltjes types were seen to be previously accomplishable within a framework of classical numerical integration techniques only by such special methods as the Runge-Kutta which requires relatively complex, special, and inefficient computer mechanization. Integration algorithm is generally accomplished in digital mechanization by combinations of (1) extrapolation type operations, i. e. linear weighings of past function values, (in this case with relatively simply realized weighings), and (2) transfer operations. In contrast to algorithms of the Runge-Kutta type, the Stieltjes algorithms developed made possible the minimum number of transfer operations (one per integral increment) and permitted maximum rate of computation. Actually all algorithms have an undesirable degree of (1) for transfer mechanizations of short bit length with respect to independent variable in precise computation of Stieltjes integral increments. The undesirability stems

from the requirement to quantize independent variables at given short bit lengths to an effective accuracy of much higher bit lengths. Hence analytical investigations were launched to determine the feasibility of removing this design problem, one which actually occurs in internal computation alone since multi-transfer design is typically only single or several bit. The result was the development of the theory of numerical Stieltjes integration in terms of "virtual variables" i.e., variables closely related to desired variables but which are involved in computations simpler than the desired variables. External inputs to a real time computer are not virtual variables. It is possible to alter integration algorithms used in input processing to generate answers in virtual variables. The major result of the investigation was the proof that there exist virtual variables which in general may be utilized in numerical integration with respect to general independent variables as though they changed linearly, i.e., like time, although the actual variable does not. In consequence virtual variable numerical Stieltjes integration may have precision corresponding to multi-transfer bit length potential using direct multi-transfer of that bit length. The generation of computer outputs involves simple transformation of the virtual variable using available data. The practical necessity of mechanizing the transformation depends on existence of high frequency variables and relatively high precision requirements. The theoretical relation of desired and virtual variables provides a further demonstration of the fundamental nature of the computation types evolved in the contract study, the realization of which has enabled the design of the first real time computer (for high frequency variable applications) capable of high precision with mechanization of modest complexity.

Another fundamental problem in incremental computation finds practical motivation for solution as a result of certain computation error characteristics rather than mechanization characteristics. It is generally recognized, and

was quantitatively established during Phase II that division in a conventional DDA is relatively highly inaccurate. Aerospace applications for near orbital speed missiles which undergo large altitude variations require precise division capability for solution of the basic navigation equation, coordinate transformation from satellite to earth coordinates, and many other important functions. For a full aerospace mission division algorithm mechanization is highly desirable to obtain the required accuracy, though a breakthrough in modified conventional DDA design (Chapter IX) presents a design alternative. To develop a division algorithm mechanization which yields high precision there must be a fundamental theory of numerical quotient computation. Such a theory was developed by transforming the results of the theory of numerical integration into numerical quotient computation form. The results of this analysis are a basis of the design of the full scale computer developed in the contract study, and have been simulated to confirm their validity.

3.1 SIMPLIFIED COMPUTATION IN TERMS OF VIRTUAL VARIABLES IN EXECUTING STIELTJES NUMERICAL INTEGRATION PROCESSES -

A. INTRODUCTION - An investigation into the theory of Stieltjes numerical integration for incremental computation, including input processing and internal computation, has led to an important combined computation structure. The relation of incremental computer processing for function generation, to over-all computer processing involving real time input variables, has been delineated in quantitative form with important implications of significantly simplified mechanization capable of precise computation for both types of computers. Generally, it has been shown that Stieltjes numerical integration can be carried out in terms of "virtual" variables, by which is meant variables not

equal to the desired variables but bearing a fixed transformation relationship to them. Proper algorithm in terms of virtual variables can be made simpler for an internal computer, or function "generation" computer, in that the classical non-Stieltjes numerical integration algorithms apply to them. However, in principle it is required that the true variables, when they are to be extracted from the computer for external use, be generated by inverse transformation of the virtual variables. Since it was shown in continuing analysis that there exists a class of virtual variables, which are slightly lagged relatively, and any one of which satisfies the simplified algorithm relationship, the specific virtual variables which most closely approximate the true variables could be chosen. Analysis indicated that the pertinent virtual variable differs from the true variable by a second difference of magnitude of $(f/IR)^2$ approximately, where f = frequency of variable and IR = iteration rate; for example if $f \leq 0.5$ cps, $IR \leq 100$ iter/sec the virtual and time variables differ by <0.003 percent. Typically, the greatest demand for accuracy in high frequency variable computations, is within the computer (where feedback effects can cause error growth), rather than in outputs. The outputs have the highest accuracy demands, in applications such as navigation, in the variables with primarily low frequency content. Thus, outputs in contrast to inputs and intermediate variables of a precision computer may be taken with usually close approximation as the virtual variables. The output mechanism of the computer system is analyzed elsewhere in the report, in relation to the contingency (for general computation in aerospace applications) of conversion by elementary transformation from virtual variables to true variables in outputs. The internal

computer design in this study is the proposed QDDA whose registers contain the information available and necessary for the virtual variable to true variable conversion in mechanization through alternative use of logic necessarily present for integrator operation in effecting the precision algorithm. The carrying out of this design modification is determined by the particular computer applications, i.e. their precision requirement and variable frequencies.

The theory of the combined input processor, internal computer algorithm structure involving virtual variable computation, leads to less complex internal computer design. In principle, an input processor designed as part of this structure should generate virtual rather than true variables. However, the following factors led to a decision to fabricate the previously proposed (strap-down) input processor design:

1. The previously proposed input processor design generates true variables most readily used to evaluate design performance.
2. A later combined input processor internal computer system has an input processor design obtainable by modest logical design modifications relative to the previously proposed unit.

- B. ANALYSIS - Consider the generating function for parametric calculation form associated with unlagged integrand and independent variables, in numerical Stieltjes integration (Chapter 5, Section 5 of the Phase I final report):

$$F(a, b) = \left[\frac{-\epsilon_{ab}}{\ln(1-\epsilon_{ab})} \right] \left[\frac{\ln(1-\epsilon_b)}{-\epsilon_b} \right] \epsilon_b \quad (3-1)$$

Denote the operators,

$$\left[\frac{-\delta_{ab}}{\ln(1 - \delta_{ab})} \right] = \theta_{ab} \quad (3-2)$$

$$\left[\frac{\ln(1 - \delta_b)}{-\delta_b} \right] = \left[\frac{-\delta_b}{\ln(1 - \delta_b)} \right]^{-1} = \theta_b^{-1} \quad (3-3)$$

the operator O with respect to any variable has inverse O^{-1} since $O \cdot O^{-1} = 1$ as seen in the case θ_b^{-1} . Then

$$F(a, b) = \theta_{ab} \theta_b^{-1} \delta_b \quad (3-4)$$

A similar relationship holds for lagged or led integrand in the parametric calculation cases, since the associated generating function for p iterations lag (p may be positive or negative),

$$F^p(a, b) \left[y_n x_n \right] = \frac{F(a, b) b^p}{(ab)^p} \left[(a^p y_n) x_n \right] \quad (3-5)$$

then

$$F^p(a, b) = \frac{F(a, b) b^p}{(ab)^p} \delta_b \quad (3-6)$$

substituting the $F(a, b)$ function

$$\begin{aligned} F^p(a, b) &= \left[\frac{-\delta_{ab}}{(ab)^p \ln(1 - \delta_{ab})} \right] \circ \left[\frac{-\delta_b}{b^p \ln(1 - \delta_b)} \right]^{-1} \delta_b \\ &= \theta_{ab}^p \circ \theta_b^{-1} \circ \delta_b \end{aligned} \quad (3-7)$$

where:

$$\theta_x^* = \frac{-\delta_x}{(x\rho \ln(1 - \delta_x))} \quad (3-8)$$

x being either ab or b . If computation of lagged integral is considered, then note from

$$\Delta I_n = F(a, b) y_n x_n \quad (3-9)$$

$$\Delta I_n^{(ab)} = \frac{F(a, b) b^{p-q}}{(ab)^{p-q}} \left[(y_n a^p) (x_n b^q) \right] \quad (3-10)$$

any p, q , from which

$$\Delta I_{n-q} = \frac{F(a, b)}{(ab)^{p-q}} b^{p-q} y_{n-p} x_{n-q} \quad (3-11)$$

The associated generating function integral and independent variable lagged q iterations and integrand lagged p iterations is

$$F^{**}_{(a, b)}(q, p) = \frac{F(a, b) b^{p-q}}{(ab)^{p-q}} \quad (3-12)$$

the superscripts on the left indicating q lagged integral and independent variable, p lagged integrand

$$\begin{aligned} F^{**}_{(a, b)} &= \left[\frac{-\delta_{ab}}{(ab)^{p-q} \ln(1 - \delta_{ab})} \right] \circ \left[\frac{-\delta_b}{b^{p-q} \ln(1 - \delta_b)} \right]^{-1} \delta_b \\ &= \theta_{ab} \circ \theta_b^{q-p-1} \circ \delta_b \end{aligned} \quad (3-13)$$

where

$$\theta_{\bar{x}}^{**}(q, p) = \left[\frac{-\delta_x}{x^{p-q} \ln(1 - \delta_x)} \right] \quad (3-14)$$

In terms of the formula for integration in these variables

$$\Delta I_{n-q} = \left[\theta_{ab}^{**} y_{n-p} \circ (\theta_b^{**})^{-1} \delta_b x_{n-q} \right] \quad (3-15)$$

Consider now a computation in terms of virtual variables of x , y and integral designed to be simpler than that of direct parametric algorithm computation. Take the virtual x_n^* variable to be x_n^* defined by

$$x_n^* = \theta_b^{**} {}^{-1} x_{n-q} \quad (3-16)$$

then

$$x_{n-q} = \theta_b^{**} x_n^* \quad (3-17)$$

Taking the virtual variable of integral to be

$$\Delta I_n^* = \theta_{ab}^{**} {}^{-1} \Delta I_{n-q} \quad (3-18)$$

then

$$\Delta I_{n-q} = \theta_{ab}^{**} \left[y_{n-p} \circ \delta_b x_n^* \right] \quad (3-19)$$

multiplied by $\theta_{ab}^{**} {}^{-1}$ yields

$$\Delta I_n^* = y_{n-p} \circ x_n^* \delta_b \quad (3-20)$$

the virtual variables I_n^* , x_n^* being formed by identical operators and delay on the desired variable,

$$()_n^* = \theta_x^{**} {}^{-1} ()_{n-q} \quad (3-21)$$

consider computation in which y variable is converted to a virtual variable of the identical kind to that of I , X , for which

$$y_{n-q} = \theta_a^{**} y_n^* \quad (3-22)$$

note

$$y_{n-p} = y_{n-q} a^{p-q} = a^{p-q} \theta_a^{**} y_n^* \quad (3-23)$$

then substituting y_{n-p} in the partially virtual variable equation

$$\Delta I_n^* = \theta_a y_n^* x_n^* \delta_b \quad (3-24)$$

where

$$\begin{aligned} \theta_a &= a^{p-q} \theta_a^{**} \\ &= a^{p-q} \left[\frac{-\delta_a}{a^{p-q} \ln(1 - \delta_a)} \right] \\ &= \left[\frac{-\delta_a}{\ln(1 - \delta_a)} \right] \\ &= \theta_a \end{aligned} \quad (3-25)$$

thus,

$$\Delta I_n^* = \theta_a y_n^* \circ x_n^* \delta_b \quad (3-26)$$

is a virtual variable algorithm involving a single operator process on integrand but not independent variable and a single whole number multiplication per iteration. The corresponding algorithm described in previous analysis as a nonparametric algorithm not

involving virtual variables involves a single operator but two or more whole number multiplications.

This virtual variable algorithm can be directly applied in any incremental function generation (without external inputs) by simply starting variables at virtual values. The generated virtual variables are at subsequent times related to the desired variables by a linear operator generally differing slightly for the actual variable. In closed loop computations the crude use of approximate algorithm could, of course, lead to a build up of large errors. But this is avoided in the virtual variable case, because only the externally observed variable is in error.

The problem of processor application of the virtual variable approach is associated with the problem in generating x_n^* where

$$\begin{aligned}
 x_n^* &= \theta_b^{**-1} x_{n-q} \\
 &= \left[\frac{-\delta_b}{b^{p-q} \ln(1 - \delta_b)} \right]^{-1} x_{n-q} \\
 &= \left[\frac{\ln(1 - \delta_b)}{-\delta_b} \right] x_{n-p} \\
 &= \left[1 + \frac{\delta_b}{2} + \frac{\delta_b^2}{3} + \dots \right] x_{n-p} \tag{3-27}
 \end{aligned}$$

assuming that an input delay of rT is reliable then

$$\begin{aligned}
 x_n^* &= \left[1 + \frac{\delta_b}{2} + \frac{\delta_b^2}{3} + \dots \right] b^{p-r} x_{n-r} \\
 &= \left[1 + \frac{\delta_b}{2} + \frac{\delta_b^2}{3} + \dots \right] (1 - \delta_b)^{p-r} x_{n-r} \\
 &= \left[1 + \frac{\delta_b}{2} + \frac{\delta_b^2}{3} + \dots \right] \left[1 - (p-r) \delta_b \right. \\
 &\quad \left. + \frac{(p-r)(p-r-1)}{2} \delta_b^2 \right] x_{n-r} \\
 &= \left[1 + \left(\frac{1}{2} - (p-r) \right) \delta_b + \left(\frac{1}{3} - \frac{(p-r)}{2} \right. \right. \\
 &\quad \left. \left. + \frac{(p-r)(p-r-1)}{2} \right) \delta_b^2 \right] x_{n-r} \tag{3-28}
 \end{aligned}$$

taking $(p-r) = 1/2$ then

$$\frac{1}{3} - \frac{(p-r)}{2} + \frac{(p-r)}{2} (p-r-1) = \frac{1}{3} - \frac{1}{4} + \frac{1}{4} \left(-\frac{1}{2} \right) = \frac{1}{24} \tag{3-29}$$

hence for $(p-r) = 1/2$,

$$x_n^* = \left[1 - \frac{1}{24} \delta_b^2 \right] x_{n-r} \tag{3-30}$$

$$x_n^* = x_{n-r} - \frac{1}{24} \Delta^2 x_{n-r} \tag{3-31}$$

is the relation of the actual desired variable to the virtual variable computed. Here x_n can be taken as ΔI_n , y_n , as well as the variable x_n of (3-1). Then the algorithm computation in terms of

virtual variables is Eq (3-2) for undelayed variables, which in direct computation form is

$$\Delta I_n^* = \left[y_n^* - \frac{1}{2} \Delta y_n^* - \frac{1}{12} \Delta^3 y_n^* \right] \Delta x_n^* \quad (3-32)$$

For undelayed and also delayed variables the classical algorithm applying to the case of uniformly increasing independent variables holds for general independent variables in virtual variables.

3.2 ANALYSIS OF HIGHER ORDER PARALLEL INTEGRATION ALGORITHM FOR COMPUTATIONS INVOLVING DIVISION - Parallel integration algorithm of a QDPU (Quotient Differential Processing Unit) was investigated for modes involving division. The sought for QDPU design has the purpose of equivalently computing in the manner (apart from round off properties) of an integrator ensemble in each of two parallel channels. The quotient generation action of the QDPU differs in fundamental form from that of a pure integration process. A theory of higher order integration algorithm for the QDPU for second order integration accuracy is of interest in an incremental computer design. The desired second order algorithm is two levels of accuracy greater than those designed in existing DDA hardware. Some investigators have formed and used the DDA algorithm problem as the resultant of a goal to accurately execute incrementation of algebraic relations and to solve difference equations inferentially obtained from the application computation. Actually the great majority of application computations involve a close relationship between both algebraic and integral incrementation. Algebraic relation incrementation is readily expressible in terms of equivalent integral incrementation, however, the reverse process is possible only through the use of appropriate integration algorithm in one manner or another. The concept of effecting accurate computation without mechanization of precision integration relies on finding a set of difference equations (ordinarily a

modification of the application differential equations) which are equivalent to the desired calculation for incremental computations. In the applications of real importance the desired calculation is almost without exception of such non-linear nature and coupling complexity that it would be a matter of luck to find the equivalent set of difference equations sought, and usually a price in computation accuracy would result. One of the two outputs of the QDPU for a mode involving an element of division has the purpose of executing the computation.

$$\Delta \theta_n = \int_{(n-1)\tau}^{n\tau} \frac{P(t)}{U(t)} dx(t) \quad (3-33)$$

The theory of Stieltjes numerical integration for virtual variables relates ΔQ_n to P_n , U_n , x_n and various differences thereof for a given order of accuracy. The QDPU mechanization effects transfers and decision processes in a manner differently than the direct algorithm form. The analytical problem of deriving explicit QDPU algorithm is that of finding the equivalent mechanizable algorithm. Analysis has led to an equivalent algorithm of second order accuracy, the terms of which are mechanizable with second difference communication with the possible exception of a single small term involving a second difference term of the independent variable. The corresponding case of ordinary integration in virtual variables did not require a second difference term in the algorithm of the independent variable, which was the purpose of the introduction of virtual variables. The implications of this result will be examined further in relation to theory, mechanization, and accuracy for the QDPU.

The theory of Stieltjes numerical integration developed in Phase I and simplified algorithms for machine computation dealt with ordinary incremental integration. Thus the operation

$$\Delta \theta_n = \int_{(n-1)\tau}^{n\tau} y \, d\chi \quad (3-34)$$

is effectively performed on the given series of y_n values given in the series of ΔX_n , by carrying out the computation in virtual variables y^* and x^* using an algorithm of form

$$\Delta \theta_n^* = [y_n^* + \lambda_1 \Delta y_n^* + \lambda_2 \Delta^2 y_n^* + \dots] \Delta X_n^* \quad (3-35)$$

The problem of incremental computation where y_n is not given but rather, where,

$$y_n = p_n/v_n \quad (3-36)$$

involves the given information Δp_n , Δv_n series and $p_o; v_o$ for QDPU operation. Substituting Eq. (3-36) in Eq. (3-35) the computation should perform in one way or another in virtual variables, the asterisk of which is hereafter dropped for brevity.

$$\Delta \theta_n = \left[\frac{p_n}{v_n} + \lambda_1 \Delta \left(\frac{p_n}{v_n} \right) + \lambda_2 \Delta^2 \left(\frac{p_n}{v_n} \right) + \dots \right] \Delta X_n \quad (3-37)$$

*Chapter 5 Part 5 First Phase Technical Documentary Report on Development of an Airborne HSDDA, H. W. Banbrook, 7 July 1961. Contract AF 33(616)-6936

The following analysis is carried out to achieve second order numerical integration accuracy. The first order difference of a quotient is determined to second order accuracy as follows:

$$\begin{aligned}\Delta(p_n/v_n) &= p_n/v_n - p_{n-1}/v_{n-1} = \frac{p_n}{v_n} \left[1 - \frac{(1 - \Delta p_n/p_n)}{(1 - \Delta v_n/v_n)} \right] \\ &\approx \frac{\Delta p_n}{v_n} - \frac{p_n}{v_n^2} \Delta v_n + \frac{\Delta p_n \Delta v_n}{v_n^2} - \frac{p_n}{v_n^3} (\Delta v_n)^2 \quad (3-38)\end{aligned}$$

For the same accuracy level when used in Eq. (3-37),

$$\begin{aligned}\Delta^2(p_n/v_n) &= \Delta(\Delta(p_n/v_n)) \approx \Delta\left(\frac{\Delta p_n}{v_n} - \frac{p_n}{v_n^2} \Delta v_n\right) \\ &\approx \Delta\left(\frac{\Delta p_n}{v_n}\right) - \Delta\left(\frac{p_n}{v_n}\right) \frac{\Delta v_n}{v_n} - \frac{p_{n-1}}{v_{n-1}} \Delta\left(\frac{\Delta v_n}{v_n}\right) \quad (3-39)\end{aligned}$$

using the exact difference formula for a product. In Eq. (3-38) if $P_n \rightarrow \Delta p_n$ we obtain $\Delta\left(\frac{\Delta p_n}{P_n}\right)$ and if $P_n \rightarrow \Delta V_n$ we obtain $D(\Delta V_n/V_n)$. Truncating all terms of higher order than second,

$$\Delta^2(p_n/v_n) \approx \frac{\Delta^2 p_n}{v_n} - \frac{2 \Delta v_n \Delta p_n}{v_n^2} + \frac{2 p_n}{v_n^3} (\Delta v_n)^2 - \frac{p_n}{v_n^2} \Delta^2 v_n \quad (3-40)$$

Substituting these results in Eq. (3-37),

$$\begin{aligned}\Delta\theta_n &= \left[\frac{P_n}{V_n} + \lambda_1 \left[\frac{\Delta P_n}{V_n} - \Delta V_n \frac{P_n}{V_n^2} \right] + \lambda_2 \frac{\Delta^2 P_n}{V_n} \right. \\ &\quad \left. + (\lambda_1 - 2\lambda_2) \frac{\Delta V_n \Delta P_n}{V_n^2} + (2\lambda_2 - \lambda_1) \frac{P_n}{V_n} (\Delta V_n)^2 \right. \\ &\quad \left. - \lambda_2 \frac{P_n}{V_n} \Delta^2 V_n \right] \Delta X_n \end{aligned} \quad (3-41)$$

The QDPU must effect Eq. (3-37) or Eq. (3-41) without using division directly. To do this assume a realizable form to QDPU computation and investigate the level of equivalence to Eq. (3-41) possible by parameter adjustment. The form

$$\Delta\theta_{n,QDPU} = \left[\frac{P_n + \lambda_1^2 \Delta P_n + \lambda_2^2 \Delta^2 P_n + \epsilon_1}{V_n + \mu_1^2 \Delta V_n + \mu_2^2 \Delta^2 V_n + \epsilon_2} \right] \Delta X_n \quad (3-42)$$

is realizable in the stated sense (assuming that ϵ_1, ϵ_2 , which are of second order are individually realizable) as is shown in later presentation the theory of QDPU operation. To put Eq. (3-42) into the form of a polynomial in $\Delta P_n, \Delta V_n$, similar to Eq. (3-41) use the relation obtained by the geometric series:

$$\begin{aligned}\left[V_n + \mu_1^2 \Delta V_n + \mu_2^2 \Delta^2 V_n + \epsilon_2 \right]^{-1} &= \frac{1}{V_n} \left[1 - \mu_1^2 \frac{\Delta V_n}{V_n} \right. \\ &\quad \left. - \mu_2^2 \frac{\Delta^2 V_n}{V_n} + \mu_1^2 \frac{\Delta V_n^2}{V_n^2} - \frac{\epsilon_2}{V_n} \right] \end{aligned} \quad (3-43)$$

Substituting Eq. (3-43) in Eq. (3-42) and expanding the product,

$$\begin{aligned}\Delta \theta_n_{QDPU} &= \left[\frac{P_n}{V_n} + \left[\lambda_1^* \Delta P_n - \mu_1^* \frac{P_n}{V_n^2} \Delta V_n \right] \right. \\ &\quad + \left[\lambda_2^* \frac{\Delta^2 P_n}{V_n} - \frac{P_n}{V_n} = \mu_2^* \Delta^2 V_n - \lambda_2^* \mu_1^* \frac{\Delta P_n \Delta V_n}{V_n^2} \right. \\ &\quad \left. \left. + \frac{P_n}{V_n} \mu_1^* \frac{\Delta V_n}{V_n^2} + \frac{\epsilon_1}{V_n} - \frac{P_n}{V_n^2} \epsilon_2 \right] \right] \Delta X_n \quad (3-44)\end{aligned}$$

For equivalence of Eq. (3-42) to Eq. (3-41) to first order, the first order terms of Eq. (3-44) must equal those of Eq. (3-41), hence take

$$\lambda_1^* = \lambda_1, \quad \mu_1^* = \lambda_1$$

The difference of the second order terms of the QDPU calculations of Eq. (3-42) from those of the desired calculation Eq. (3-41) is

$$\begin{aligned}&\left[(\lambda_2^* - \lambda_2) \frac{\Delta^2 P_n}{V_n} + (\lambda_2 - \mu_2^*) \frac{P_n \Delta^2 V_n}{V_n^2} + \frac{\epsilon_1}{V_n} - \frac{P_n}{V_n^2} \epsilon_2 \right. \\ &\quad \left. + (\lambda_1^2 + \lambda_2 - 2\lambda_2) \left(\frac{P_n}{V_n} \Delta V_n - \frac{\Delta P_n}{V_n} \right) \frac{\Delta V_n}{V_n} \right] \Delta X_n \quad (3-45)\end{aligned}$$

The first two terms may be nulled taking

$$\begin{aligned}\lambda_2^* &= \lambda_2 \\ \mu_2^* &= \lambda_2\end{aligned}$$

The last term is predetermined by the first order equivalence conditions.

The error of the QDPU algorithm with $\lambda_1^* = \mu_{1n}^* \lambda_1$, $\lambda_2^* = \mu_{2n}^* = \lambda_2$ is:

$$\begin{aligned} \epsilon_{\Delta\theta_1, \text{QDPU}} &= (\lambda_1^* + \lambda_1 - 2\lambda_2) \frac{\Delta V_n}{V_n} \left(\frac{P_n}{V_n^2} \Delta V_n - \frac{\Delta P_n}{V_n} \right) \\ &\quad - \left(\frac{P_n}{V_n^2} e_2 - \frac{e_1}{V_n} \right) \Delta X_n \end{aligned} \quad (3-46)$$

Such realizable choices of e_1 , e_2 , are sought so that the error is minimized. First consider the explicit algorithms Eq. (3-35) sought for realization noting that the QDPU algorithm has analogous form

$$\Delta\theta_n, \text{QDPU} = \left[\frac{P_n + \lambda_1 \Delta P_n + \lambda_2 \Delta^2 P_n + e_1}{V_n + \lambda_1 \Delta V_n + \lambda_2 \Delta^2 V_n + e_2} \right] \Delta X_n \quad (3-47)$$

except for e_1 , e_2 , to be determined.

For unlagged input variables $\lambda_1 = -1/2$, $\lambda_2 = -1/12$, hence the constant coefficient of the error term Eq. (3-46) is

$$(\lambda_1^* + \lambda_1 - 2\lambda_2) = -1/12$$

For lagged input variables $\lambda_1 = +1/2$, $\lambda_2 = +5/12$ and

$$(\lambda_1^* + \lambda_1 - 2\lambda_2) = -1/12$$

the same value. When only one variable is lagged it is readily shown that Eq. (3-47) is replaced with an expression obtained by replacing the lagged variable terms with that obtained by the substitution

$$X_n \longrightarrow X_n + \Delta X_n + \Delta^2 X_n \quad (3-48)$$

the choice of ϵ_1 , ϵ_2 being unaltered because of their second order magnitude. Thus the choice of ϵ_1 , ϵ_2 is generally that which minimizes

$\epsilon_{\Delta\theta_{nQDPU}}$ where

$$\begin{aligned}\epsilon_{\Delta\theta_{QDPU}} &= -\frac{1}{12} \frac{\Delta V_n}{V_n} \left(\frac{P_n}{V_n^2} \Delta V_n - \frac{\Delta P_n}{V_n} \right) \Delta X_n \\ &\quad - \left(\frac{P_n}{V_n^2} - \epsilon_2 - \frac{\epsilon_1}{V_n} \right) \Delta X_n\end{aligned}\tag{3-49}$$

Consider choices ϵ_1 , ϵ_2 and their implications. If we choose

$$\epsilon_1 = \frac{-\Delta V_n}{12} \Delta(P_n/V_n)\tag{3-50}$$

$$\epsilon_2 = 0$$

then $\epsilon_{\Delta\theta_{QDPU}} = 0$ formally. The ϵ_1 term enters as a small second order term when the quotient P_n/V_n changes in a regular manner e.g. where $V_n \neq 0$. No satisfactory way of effecting the term generally in an incremental algorithm has been devised. The inclusion of the term in special cases will be discussed later. Approximate second order algorithm is given with ϵ_1 , $\epsilon_2 = 0$ in Eq. (3-49). Write this relation (which can be derived readily by approximate analyses) in the form

$$0 = \tilde{P}_n \Delta X_n - \tilde{V}_n \Delta \theta_n$$

where

$$(\tilde{})_n = (\)_n + \lambda_1 \Delta (\)_n + \lambda_2 \Delta^2 (\).$$

For the moment assume that $\Delta\theta_n$ is estimated in some manner and that an R register is updated according to

$$\bar{R}_n = \bar{R}_{n-1} + \tilde{P}_n \Delta X_n - V_n \Delta \theta_n\tag{3-51}$$

Then if $\Delta\theta_n$ had been estimated exactly (according to the approximate second order level) then no net change would occur in \bar{R} in that iteration. A residue of past errors is reflected in \bar{R} on the value \bar{R}_{n-1} . Consider the method of making estimates of $\Delta\theta_n$ in the case where $\Delta\theta_n$ is represented as a single increment. The $\Delta\theta$ value actually available is the lagged value estimated at the previous iteration. Let the decision that output $\Delta\theta_n$ be represented as a given magnitude be made using the magnitude of R_n where,

$$R_n = \bar{R}_{n-1} + \tilde{P}_n \Delta X_n \quad (3-52)$$

where \bar{R}_{n-1} is a residue corrected for all past decisions by appropriate addition of $-\hat{V} \Delta\theta$ terms; accordingly the relation

$$\begin{aligned} \bar{R}_{n-1} &= R_{n-1} - \tilde{V}_{n-1} \Delta\theta_{n-1} \\ &= R_{n-1} - \tilde{V}_{n-1} \Delta\theta_n^L \end{aligned} \quad (3-53)$$

where $\Delta\theta_n^L$ is the lagged incoming value of the nth iteration equal to the computed value of the $(n-1)^{st}$ iteration. Then the computation of R_n is given by

$$R_n = R_{n-1} + \tilde{P}_n \Delta X_n - \tilde{V}_{n-1} \Delta\theta_n^L \quad (3-54)$$

The decision for output $\Delta\theta_n$ (no overflow) is according to a correction which minimizes the absolute value of $R_n - \tilde{V}_n \Delta\theta_n$.

In multi-increment computation use may be made of the relation

$$\Delta\theta_n = \Delta\theta_{n-1} + \Delta^2\theta_n \quad (3-55)$$

where $\Delta\theta_{n-1}$ is known and $\Delta^2\theta_n$ is to be determined. Then the ideal R-register value is

$$\begin{aligned}\bar{R}_n &= \bar{R}_{n-1} + \tilde{P}_n \Delta X_n - \tilde{V}_n \Delta \theta_n \\ &= \bar{R}_{n-1} + \tilde{P}_n \Delta X_n - \tilde{V}_n \Delta \theta_{n-1} - \tilde{V}_n \Delta^2 \theta_n\end{aligned}\quad (3-56)$$

Making the decision as to the magnitude of $\Delta^2 \theta_n$ we may use the magnitude R_n given by

$$R_n = \bar{R}_{n-1} + \tilde{P}_n \Delta X_n - \tilde{V}_n \Delta \theta_{n-1} \quad (3-57)$$

where

$$\bar{R}_{n-1} = R_{n-1} - \tilde{V}_{n-1} \Delta^2 \theta_{n-1} \cdot \bar{R}_{n-1}$$

being the ideal of the past iteration which at the next iteration is realizable. Thus multi increment quotient algorithm determines output $\Delta^2 \theta_n$ according to magnitude

$$R_n = R_{n-1} + \tilde{P}_n \Delta X_n - \tilde{V}_n \Delta \theta_n^L - \tilde{V}_{n-1} \Delta^2 \theta_n^L \quad (3-58)$$

where

$$\Delta \theta_n^L = \Delta \theta_{n-1}, \Delta^2 \theta_n^L = \Delta^2 \theta_{n-1},$$

the subscript indicating lagged outputs which are inputs to the computation. The decision for output $\Delta^2 \theta_n$ is made according to the choice which minimizes the absolute value of

$$R_n - \tilde{V}_n \Delta^2 \theta_n$$

The computations for different digital representations are presented in chapters VII and VIII.

Consider again the problem of the (small) second order term ϵ_1 omitted in the algorithm just derived. If formally included the more exact calculation of R_n is

$$R_n = R_{n-1} + \left[\tilde{P}_n + \epsilon_1 \right] \Delta X_n - \tilde{V}_n \Delta \theta_n^L - \tilde{V}_{n-1} \Delta^2 \theta_n^L$$

where

$$\begin{aligned}\epsilon_1 \Delta X_n &= \frac{-\Delta V_n}{12} \Delta \frac{P_n}{V_n} \Delta X_n \\&= \frac{-\Delta V_n}{12} \Delta \frac{P_n}{V_n} \Delta X_n - \frac{P_n}{V_n} \Delta^2 X_n \\&= \frac{-\Delta V_n}{12} \Delta^2 \theta_n + \frac{\Delta V_n \Delta^2 X_n}{12} - \frac{P_n}{V_n}\end{aligned}$$

If the primary contribution is from the first of the two terms of $\epsilon_1 \Delta X_n$ then improved calculation is

$$R_n = R_{n-1} + \tilde{P}_n \Delta X_n - \tilde{X}_n \Delta \theta_n^L - \tilde{V}_{n-1} + \frac{\Delta V_n}{12} \Delta^2 \theta_n^L$$

for approximate second order algorithm.

CHAPTER IV
**DESIGN CONCEPTS FOR COMPUTER SYSTEMS WITH
PROGRAMMABLE INPUT PROCESSING CAPABILITY**

4.0 INTRODUCTION - Input processing is required in special computation routines demanding a level of computation capability higher than the internal computer processes, which have already been assigned the time consuming computation routines associated with the computation task of the mission. The special computation routines requiring input processing are characterized by one or both of the following:

1. High frequency inputs presenting rate handling and precision problems to an incremental computer.
2. Bulky data processing at high rate, the structure of which is simple and repetitive in form.

A hybrid GP-DDA computer system without the input processor mode is capable of performing moderately demanding computation tasks which could not be performed by a less sophisticated system. Many special routines of an application computation are sufficiently demanding as to require full input processing in addition to the normal processing capability of the direct GP-DDA combination. They impose their input processing requirement as a result of a need for special digital processing features which (as a major result of this phase of the contract study) are shown to be largely attainable (without significant increase in complexity) by implementing share modal action* of the GP-QDDA system or QDDA computer to achieve input processing

*The processing action resulting from essentially the same hardware complex used through instantaneous modal switching to effect modified multiplier bit length or modified subroutine iteration rate.

functions. There are four design problems involved in realizing programmable input processing by share modal action. These problems require:

1. Multiplier share and modal switching action which implement input processing without significant increase in system complexity.
2. Programmability of input processing routine to perform any of a number of input processing applications such as: strap-down computations, midcourse guidance, air data computations during re-entry for ICBM terminal guidance, damping and digital servo computations in navigation, digital autopilot, fire control, and radar terrain picturing.
3. Programmable input pre-processing and extrapolation processing which characterize input processors of maximized iteration rate, and that achieve precision algorithm. (They present a problem in only one of the two design types developed.)
4. Communication and storage modifications for precision computation and processing versatility.

These problems will be analyzed separately in the following discussions after which results will be combined to form system configurations augmented to attain input processing for the GP-QDDA and QDDA systems.

4.1 ARITHMETIC CAPABILITY FOR INPUT PROCESSING IMPLEMENTED WITHOUT SIGNIFICANT INCREASE IN SYSTEM COMPLEXITY

- A. GP-QDDA Computer System With Input Processor - The majority of input processing computations involve high frequency variables which at moderate iteration rates must be executed with whole word multiplication or many-bit transfer. In the case of the GP-QDDA system, the whole word multiplier of the GP can

basically supply this required whole word multiplication capacity at adequate iteration rate for input processing provided balanced time sharing is achieved. The most demanding application, i. e., strap-down navigation, requires an estimated one-fourth time share of the fast multiplier services to realize adequate accuracy; other applications involving precision input processing require considerably less.

The effectively lower iteration of the GP program is more than offset by the reduced program task of the GP as a result of increased program allocation to the QDDA and input processor mode. The sharing pattern adopted for the whole word multiplier should have a time schedule which requires minimum buffering of inputs for input processing action, and minimum loss of GP program efficiency resulting from share action. For accurate incremental integration algorithm to be obtained with minimum complexity it is necessary that inputs be sampled at equal time intervals. The implementation of constant input pre-processing intervals and minimum buffering implies the adoption of a share allocation of the multiplier unit consisting of evenly spaced periods of product formation from inputs and programmed quantities. Use of a "slow" multiplier would introduce preliminary system design problems as a result of the variable time for instruction execution. In this case, to ensure that the slow multiplier is available at fixed word times, a certain amount of programming inflexibility and consequent inefficiency would be forced if the interval of continuous GP mode were very short (the latter can be avoided). To ensure essentially unimpaired GP program efficiency without providing temporary storage of GP instruction and data words at an interrupt time, the input data would be buffered the appropriate

few word times required to complete the GP instruction. Thereafter, the input processing multiplications would be called and executed with products entering a buffer to the extrapolation unit. The rather expensive buffer requirements in a system with a slow multiplier are obviated by assigning the system a more expensive multiplier, a fast multiplier such as that in the strap-down computer fabricated during Phase II of the program. The fast multiplier yields real performance improvement and is therefore appropriate in the GP-QDDA system with input processing capability obtained by share of the whole word multiplier unit.

- B. QDDA With Input Processing Capability - The extra multi-transfer bit length required for input processing can be implemented, in contrast to the previously discussed design approach by modal action of the QDDA, without significantly adding to the transfer hardware of the plain QDDA not designed to have the proposed modal action. This result is the general consequence of the mechanization of the QDDA with several multi-bit transfer units. This set of multi-transfer units can be moded to achieve longer word multiplier by modal logic of modest complexity. No timing problem exists in bringing in pre-processed inputs to the QDPU or distributing outputs since the allocation of each QDPU is programmable. The inclusion of this programmable input processing feature in the QDDA will enable the QDDA to handle any one of the word set of input processing problems with accuracy consistent with that of existing sensors associated with each application. The use of the fast multiplier of a GP-QDDA system will probably not be necessary from the standpoint of input processing word length requirement until design breakthroughs in sensor accuracy are made some time in the future.

4.2 COMMUNICATION AND PROGRAMMABILITY OF WHOLE WORD INPUT PROCESSING IN THE GP-QDDA SYSTEM - Programmability of GP and QDDA resolves most problems of implementing programmable input processing. Those programmability problems which do occur are for the GP-QDDA system which effects input processing using the whole word multiplier of the GP. These problems are associated with input pre-processing and extrapolation operations, which may be implemented in relatively simple parallel processing loops, to achieve high precision integration algorithm, at maximum real iteration rate of input processing, yet leaving adequate processing time for ordinary GP operation.

The problem of designing the pre-processing and extrapolation processor loops is complicated by the requirement to handle any of a wide variety of input processor applications involving different numbers of inputs, and different numbers of operations on each input. The input processing routine, which is executed in $N_I \leq 2^k$ word times*, is blended together with the residual GP program in such manner that up to one-fourth of all word times are allocated to the input processing. This is accomplished as follows: The normal procedures of GP programming are employed, the input processor routine is programmed at the start, then $(4 \cdot 2^k - N_I)$ words of ordinary GP program, then the input processor routine, and so on until the entire GP program is complete. The pre-processing loop makes available a particular pre-processed input at a given word time at modulo 2^k . Assuming the number of inputs is ≤ 4 , then input accumulation and partial extrapolation quantities can be updated and stored in six circulating lines of two words on a drum which make available without delay the pre-processed inputs for subsequent processing by the fast multiplier.

*Here k is the least integer such that 2^k is greater than the number of words in the input processing routine.

of the GP. During each input processing phase the multiplier outputs are fed directly to the extrapolator unit. In consequence of the relatively long period until updating of the next input processing, which is more than four times the input processing period, the extrapolator unit need not be constrained to have its contents available on short notice. Assuming that the shortest routine encountered in input processing applications is eight word times, then the extrapolation unit may take 32 word times without presenting a timing problem, an amenable relationship for input processings of ≤ 32 word routines (strap-down calculations have a 27 word routine). The extrapolator has three 32 word circulating lines which have simple logic for extrapolator processing in the same manner as the strap-down processor. The programming structure of the GP may be retained in this system by adding an input processing state counter, which during input processing periods, signals the transfer of pre-processed inputs to, and extrapolator inputs from, the fast multiplier rather than according to address. The extrapolator unit has outputs which are increments used for updating the final outputs of the input processing operation. The problem of updating and communicating the final input processing outputs may be resolved by mechanizing special parallel updating logic for 32 words of the rapid access memory. The increment quantity outputs from the extrapolator are associated with these 32 words and automatically update these words in fixed order during proper state of the input processing counter. A GP computer with state of the art word rates can be given input processing capability with complete adequacy at 100 to 200 it/sec and yet retain more than 75 percent of computation capacity for ordinary GP operation.

4.3 COMMUNICATION AND STORAGE MODIFICATIONS OF QDDA FOR PROGRAMMABLE INPUT PROCESSING CAPABILITY - The programmable QDDA with 2M bit transfer for each of the two parallel channels obtainable by modal switching of pairs of M bit transfer units during the first ω word times ($\omega \leq 16$) generates multi-increment outputs of 2M bits representing first

differences. According to mechanization of register memory by cores or drum there must be provision for the added communication requirements of 64M bits. Core registers require increased communication wiring, and drum registers require additional rapid access memory bits (both by a count of 64M). For M = 4 the QDDA is capable of all input processing functions including strap-down navigation with accuracy consistent with state of the art sensors. A multi-increment QDDA with 128 QDPU ordinarily requiring 512 bits of core memory for the drum register case, requires 768 bits for programmable input processing capability in 16 QDPU with eight bit transfer, and with a minimum of 112 QDPU with four bit transfer for the remaining computation task of the mission.

CHAPTER V
EVALUATION OF AUXILIARY DDA DESIGN
AND COMPUTATION FEATURES

5.0 INTRODUCTION - A number of possible design features of DDA which generally effect programming ease, computation versatility and capacity in substantial but limited degree, are described and evaluated. As will be seen the more valuable of these auxiliary features are incorporated in the full scale computer system design which is the major product of this study. The design features evaluated are:

1. Variable (Programmable) Word Length
2. Output and Multi-Input Scaling Programmability (of the type 2^{-K} , K Integral)
3. Multi-Input Quantization
4. Decision Operation
5. Communication Programmability
6. Derivary Communication^{*}
7. Servo Operation

Also, a number of pertinent computation features are analyzed.

5.1 VARIABLE (PROGRAMMABLE) WORD LENGTH - Different portions of the computation program generally have markedly different accuracy requirements and involve computation variables with markedly different maximum rates. A computer with variable word length can be programmed to handle these computations with a substantial net saving in program bit length, compared to that of a DDA with fixed word length (in which portions of a majority

*See First Phase Technical Documentary Report, pp 15-19.

of the registers are not used). Depending on the application the resultant iteration rate of the variable word length DDA may be 30 to 75 percent greater than that of the fixed word length DDA. It has been found in programming a DDA with incomplete communication (one integrator not being capable of picking up outputs of any chosen integrator, but perhaps any one of half the total set) that full exploitation of variable word length for increased iteration rate is not generally possible, and that perhaps half the gain is realized. There are other design conditions which can reduce the full advantage somewhat, such as minimum word length imposed when parallel lines on a drum are used for storage of information. On the whole, mechanization for variable word length and concomitant advantages of input and output scaling are among the cheapest significant gains in computation capability of a sophisticated incremental computer.

5.2 OUTPUT AND MULTI-INPUT SCALING PROGRAMMABILITY -
DDA computation accuracy is highest when the maximum output rate of the R register approaches unity. It is therefore of very real value to be able to adjust the general order of magnitude of computation variables by scaling in a simple manner which does not require scaling integrators. Since in a delay line containing a binary number an added k bit delay is equivalent to a 2^k scale change, the mechanism for achieving input and output scale changes is relatively simple. Output scaling has, from the standpoint of accuracy of the output, only one best scale, however, since there would be occasions when input scaling cannot be arranged to do the whole 2^k scaling of variables it can have some practical value in cases where the input scaling range is limited. For variable word length DDA the output scaling is free. Since a single output can be required at a number of inputs with different scales it is clear that adequate 2^k scaling cannot be done simply with output scaling but also requires input scaling. The degree and kind of input scaling capability can have substantial effect on the ultimate computation capacity obtained through increased

programming versatility (up to a sharp limit). However, mechanization complexity could be the negative factor. Programming versatility for most applications essentially reaches the upper limit by providing for about five independent inputs to a register with independent scales. A scaling range of 2^0 to 2^8 does not make all programs directly programmable but with programming ingenuity most problems may be programmed within this degree of flexibility without loss of program efficiency or computation accuracy. The mechanization of multi-input pickup and accumulation which directly uses the delay-scale property is simplest if all inputs have a different scale since, for single increment communication only one adder is required. The capability of handling two inputs of the same scale (and afterwards inputs of at most two of the same scale) appears, however, necessary as well as sufficient for adequate programming flexibility.

The operation of input accumulation for y registers must be carried out continuously to the end of the word (in order to accomplish full updating with possible non-zero carry to the most significant end of the register). Thus a DDA with more than one y register must have individual updating arithmetic units for each y register. Two parallel DDA computers with equal computation capacity, but with a different number of y registers, involve different costs in providing required updating arithmetic units. For a given level of computation capacity, the fewer the number of y registers the better. The second feature of multi-input processing associated with algorithm is discussed in the next section.

5.3 MULTI-INPUT QUANTIZATION FOR SINGLE OR MULTI-TRANSFER -
A DDA may be designed to have multi-input programmability of (independent variable) ΔX registers (used for single or multi-transfer control). The design problem of providing for multi-inputs to ΔX registers differs from that for y registers in an important way. A difference arises from the general fact

that ΔX registers may be short as a result of few bit multi-transfer (in the case of simplest transfer mechanization the amount shorter may be the transferred y word length). Inputs are necessarily scaled to be consistent with limited transfer capability. Since ΔX registers are short their full updating could be executed usually in a fraction of a word time, implying that a single updating arithmetic unit can, in principle, (by serial sub-word operation) update several ΔX registers in a computer designed to execute a number of transfer operations in parallel. In this respect updating ΔX registers may be economical in a DDA with sophisticated processing capability. A second feature of multi-input processing for a register is the requirement for a quantization operation i. e., roundoff operation. In the case of ΔX variables the limited multi-transfer capability provided for by the mechanization requires that accumulated inputs when used for transfer control be rounded off. In the case of y variables the phasing of transfer start (depending on mechanization) in effecting integration algorithm may require a roundoff operation in generating transferred variables. Roundoff operations must be selected to remove bias. For ternary or ordinary multi-increment communication the subsignificant register quantity for multi-inputs subjected to roundoff may be initialized (at one-half) in the conventional manner for a ternary R register. The extreme shortness of multi-input registers requires that a further refinement be mechanized since such a register is biased to the extent of 2^{-N-1} parts of the maximum content where 2^{-N} is the scale of the lowest scale input. Using the sophisticated roundoff operation (simply) mechanized in the strap-down processor for input accumulator outputs to the multiplier, the bias is removed.

5.4 DECISION OPERATIONS - The essential decision capabilities recognized for DDA design are accepted here as well as elaborations important for extended DDA application. Their implementation in the full scale computer system with a remarkably new processing structure is described in Chapters X and XI.

5.5 COMMUNICATION PROGRAMMABILITY - Program versatility of a DDA is reduced if the level of communication programmability is not sufficiently high (despite use of ingenuity by the experienced programmer). A computer for a full aerospace mission may require more than four or five times the program of a computer for a primarily airborne inertial navigation function. Moreover, the majority of aerospace program subroutines may require a high degree of intercommunication. This would appear to imply that a computer for a full aerospace mission should have a higher degree of communication programmability than the conventional DDA. It will be shown in later chapters that the design approach developed for the full scale aerospace computer with multi-increment computation leads to a reduced number of computer outputs (by a factor of two) which are single increment rather than multi-increment. Consequently the mechanization required to make outputs available for rapid access as inputs is not only simpler than that of a multi-increment DDA but also simpler than that of single increment DDA. A net advantage in simplicity of total communication structure for full communication of the full scale aerospace computer, relative to the conventional DDA with equal program, is retained after further taking into account that three (or four) input variables rather than two input variables are allocated component input variables, and that the total number which must be provided is comfortably the same (six). Communication structure for partial communication, such as using z lines on a drum, appears somewhat simpler for the conventional DDA but inflicts a program capacity reduction which is significant especially for a variable word length DDA. In the case of a sophisticated incremental computer where outputs are collected in a rapid access memory, and selected by drum stored input selection words, certainly if some cramping in drum storage of input selection word set appeared economical, a certain fraction of the total number of words in the set for the generalized integrator could be shortened without appreciable communication loss. Since adequate storage space (assuming drum memory), is available

because of a provision for a set of short registers, the latter need not be re-sorted to in the proposed aerospace computer.

5.6 DERIVARY COMMUNICATION - A new type of communication (termed derivary) using the ternary set +1, -1, -0 was proposed during Phase I as a means of communicating second difference information for higher order integration without actually increasing the bits communicated. Since second order algorithms involve the factor one-third it was seen that this factor (unamenable to delay-scale mechanization) could be effectively made on second differences by counting modulo three and communicating in binary + or - when the first difference ternary is zero. The primary value of second order algorithm has been shown, however, to arise in high frequency variable computation with multi-increment accuracy, the latter because roundoff effects are reduced to below second order algorithm magnitude levels. The derivary communication concept can be applied (if required accuracy warrants) to a multi-increment computer of the revolutionary type discussed in latter chapters and which uses second difference communication of ternary type. In this case the utilization of the information space when the increment is zero provided by + and - selection, may communicate second differences with scale one-third for the integration algorithm.

5.7 PHILOSOPHY OF INTEGRATION AND QUOTIENT ALGORITHM PROCESSES IN RELATION TO SERVO ACTION IN GENERAL DDA COMPUTATION - An investigation was initiated to evaluate the degree of generality solely* of incremental integration and quotient algorithm processes alone, for computation tasks other than slewing typically assigned to an incremental computer, and tasks envisioned in future applications. The conventional DDA with codable servo operation is the excepted case in point, for the general impression has

*Decision functions however being incorporated in non-analytical processes of switching.

existed on the part of many that high precision is obtainable in some computations using implicit computation with servo loops*. At this point, the relationship of general servo mechanism function to direct computation function is one formative basis in evaluating the alternative designs. Generally, the direct computation function if precisely executed leads to accurate results, but if subject to inexact execution through integration algorithm error or roundoff error, it can lead to inaccurate non-self-correcting results. In general, the servo mechanism function is ideal to insure the reduction of large induced errors, assuming the errors have some magnitude bounds, but this is accomplished at the price of tolerating a certain range or dead zone of error inherent to servomechanism action. The largest computation execution errors in whole word incremental computations are typically those of integration algorithms. Conventional DDA design is such that roundoff error is equally as important as integration algorithm error. With the exception of applications basically requiring decision modes, simulations have shown that high precision is uniformly attained in sophisticated DDA systems, without servo elements, thus avoiding limitations in computation accuracy implied by servo mechanism computation. A special error source of the servo despite multi-input scaling is the effective lag produced by chance superposition of "1"s. Though this lag source is reduced by reducing scale to extremely small values the result is noisy servo action.

5.8 WHOLE WORD DERIVATIVE COMPUTATION IN A DDA WHICH IS LAG FREE - A variable x is updated in a DDA with single or few bit increments Δx which represent the true derivative of x . An application may require a computer output of a whole word representation of the derivative $\overset{\circ}{x}$ in which case the Δx form is inadequate from the standpoint of accuracy.

*Systems of integrators involving at least one operational integrator, i.e. servo.

The conventional methods of generating a whole word $\overset{0}{x}$ are based on linear smoothing methods involving pure integration and in addition alternatively using a servo element. The general impression that the former method must involve error as a result of a lag for band limited inputs is the result of use of unsophisticated smoothing computations, and will be shown to be a false one. Thus in effect it will also be shown that there is no natural limitation of integration methods in a DDA of this type. The importance of lag effects in a computer output variable cannot be overstated, as for example, in the applications to craft control where instability can result. Thus, in general, it is better to tolerate a higher noise level than a lag in a variable in certain critical cases. Through design effecting higher iteration rate and multi-increment computation, the noise level is greatly reduced. Thus the development of a lag free derivative computation complements overall performance improvement for such applications. There are applications in which the rates of variation of derivative variables to the output are so pronounced that the freeness of lag implied by system requirements must be such that a locally quadratic $\overset{0}{x}$ can be transmitted without lag in contrast to the less demanding case of a locally linear $\overset{0}{x}$. Consider an attack on this demanding problem through analysis using continuous linear smoothing theory. Conventional first order smoothing is given by

$$\overset{0}{\tilde{x}} = \int_{-\infty}^t K e^{-K(t-x)} \overset{0}{x} dx \quad (V-1)$$

which may be shown for $\overset{0}{x} = (\overset{0}{x})_o + (\overset{0}{x})_o t$ to introduce a lag of time $1/K$ in the $\overset{0}{x}$ generated. Computation of unlagged $x(t)$ through use of past history properties at time x in the case where no smoothing is required would be possible by Taylor series, thus

$$\overset{0}{x}(t) = \overset{0}{x}(x) + \overset{00}{x}(x)(t-x) + \overset{000}{x}(x) \frac{(t-x)^2}{2} \quad (V-2)$$

Where smoothing is obviously required over a wide range of times this suggests a general computation form which is unlagged for locally quadratic $\overset{\circ}{x}$, namely

$$\overset{\approx}{\overset{\circ}{x}} = \int_{-\infty}^t \left[\overset{\circ}{x}(x) + \overset{\circ}{x}'(x)(t-x) + \overset{\circ}{x}''(x) \frac{(t-x)^2}{2} \right] g(t-x) dx \quad (V-3)$$

where the weighting function g must satisfy

$$\int_0^\infty g(x) dx = 1$$

since substituting for the truncated Taylor series, $\overset{\circ}{x}(t)$ yields

$$\overset{\approx}{\overset{\circ}{x}} = \int_{-\infty}^t \overset{\circ}{x}(t) g(t-x) dx = \overset{\circ}{x}(t) \int_0^\infty g(x) dx \quad (V-4)$$

and we require $\overset{\approx}{\overset{\circ}{x}}(t) = \overset{\circ}{x}(t)$. The actual information available is $\Delta x(x)$, hence the linear smoothing form will be expressed in terms of $x(x)$ by transformation obtained by successive integration by parts, namely

$$\overset{\approx}{\overset{\circ}{x}} = \int_{-\infty}^t \left\{ g(t-x) - \frac{d}{dx} \left[(t-x) g(t-x) - \frac{d}{dx} \left[\frac{(t-x)^2}{2} g(t-x) \right] \right] \right\} \Delta x dx \quad (V-5)$$

Consider the selection of $g(x) = K e^{-Kx}$ for the weighing function in the second order lag free computation form.

It is shown by direct differentiation of the terms in the integrand for this case that

$$\overset{\approx}{\overset{\circ}{x}} = \int_{-\infty}^t K e^{-K(t-x)} \left[3 - 3K(t-x) + \frac{K^2(t-x)^2}{2} \right] \overset{\circ}{x} dx \quad (V-6)$$

The DDA will effect the computation by solving the differential equation obtained from the integral form. To obtain the computation in a set of first order differential equations each one of which is effected by a single integration, differentiate the integral form of x as follows

$$\frac{d}{dt} \left[\frac{d}{dt} (\tilde{x} e^{Kt}) - 3K \tilde{x} e^{Kt} \right] = \frac{d}{dt} \left[\int_{-\infty}^t e^{Kx} K [-3K + K^2(t-x)] \tilde{x} dx \right] \quad (V-7)$$

to obtain the relation

$$\frac{d}{dt} \left[\left(\frac{d\tilde{x}}{dt} + K(\tilde{x} - 3\tilde{x}) \right) e^{Kt} \right] = -3K^2 \tilde{x} e^{Kt} + K^2 \int_{-\infty}^t e^{Kx} K \tilde{x} dx \quad (V-8)$$

denoting

$$v = \frac{1}{K} \left[\frac{d\tilde{x}}{dt} + K\tilde{x} - 3K\tilde{x} \right] \quad (V-9)$$

and using

$$\tilde{\tilde{x}} = K \int_{-\infty}^t e^{-K(t-x)} \tilde{x} dx \quad (V-10)$$

$$\frac{dv}{dt} = K \left[\tilde{\tilde{x}} - 3\tilde{x} - v \right] \quad (V-11)$$

$$\frac{d\tilde{x}}{dt} = K \left[\tilde{x} - \tilde{\tilde{x}} \right] \quad (V-12)$$

From the definition v a third differential equation is obtained. Thus $\tilde{\tilde{x}}$ can be computed with three integrators each solving one of the differential equations

$$\frac{d\tilde{\tilde{x}}}{dt} = K \left[\tilde{x} - \tilde{\tilde{x}} \right] \quad (V-13)$$

$$\frac{dv}{dt} = K \left[\tilde{x}_x^o - 3\tilde{x}_x^o - v \right] \quad (V-14)$$

$$\frac{\tilde{x}_x^o}{dt} \approx K \left[v - \tilde{x}_x^o + 3\tilde{x}_x^o \right] \quad (V-15)$$

which in difference equation form are approximated by

$$\Delta(\tilde{x}_x^o) \approx (K\tau) \left[\Delta x_x - \tilde{x}_{x-1}^o \right] \quad (V-16)$$

$$\Delta(v_x) \approx (K\tau) \left[\tilde{t}^o - 3\Delta x_x - v_{x-1}^o \right] \quad (V-17)$$

$$\Delta(\tilde{x}_x^o) \approx (K\tau) \left[v_x^o - \tilde{x}_{x-1}^o + 3\Delta x_x \right] \quad (V-18)$$

which with refined algorithm yield a lag free whole word derivative \tilde{x}_x^o . The price of removing lags in the derivative estimate is some increase in noise. It may be shown that the factor of increase relative to the first order computation is given by

$$\frac{\sigma_{\text{Noise } \tilde{x}_x^o}}{\sigma_{\text{Noise } \tilde{x}_x^o}} = \left[\frac{\int_{-\infty}^{+\infty} K^2 e^{-2K(t-x)} \left[3 - 3K(t-x) + \frac{K^2}{2}(t-x)^2 \right]^2 dx}{\int_{-\infty}^{+\infty} K^2 e^{-2K(t-x)} dx} \right]^{\frac{1}{2}} = \sqrt{\frac{33K}{16}} \approx 2 \quad (V-19)$$

Thus the noise error of \tilde{x}_x^o is twice that of \tilde{x}_x^o , a surprisingly small increase in view of removal of the much more critical lag effects.

CHAPTER VI

DEVELOPMENT OF SERIAL-PARALLEL DDA MECHANIZATIONS WITH HIGH DUTY FACTORS WHICH ARE CAPABLE OF QUOTIENT ALGORITHM WITH DERIVED (SINGLE INCREMENT) TERNARY AND LATER DEVELOPED MULTI-INCREMENT COMPUTATION

6.0 INTRODUCTION - The demanding computation capability requirements for aerospace applications established in application studies and computer type computation capability analyses (See Chapter XI) demonstrated that two types of computation (a) input processing and (b) quotient operations in internal computation present overwhelming computation tasks for conventional DDA mechanization.* The necessity for and the nature of special design features for type (a) was established in Phase I of the study. The full significance of type (b) was not evolved until Phase II during which aerospace applications were analyzed in a detailed study and it was established that there were severe accuracy limitations of conventional DDA mechanizations in executing quotient type computations which incontrovertibly occur in aerospace applications. The accuracy limitations were demonstrated in both analyses and simulations. The historical role of quotient type computations is discussed more fully in the next section. The conclusion based on analysis and simulation was that an aerospace computer for a full mission must have radically different mechanization for even internal computations in consequence of quotient computation requirements. As will be seen, the ultimate design developed for a full scale computer system which executes input processing and internal computations together in a highly efficient mechanization did not reflect an overall mechanization complexity substantially greater than that required for input processing. As a result the developments in increased

*At conventional or relatively high iteration rates.

internal computation capability made possible by highly unconventional processes developed in the chapter on multi-increment QDDA are essentially gains without genuine overall mechanization cost given input processing is established to be necessary. In the analyses of this chapter the balance of factors of computation capability and mechanization complexity are examined in detail for the internal computation. The analysis is carried out for the most part as for a separate computer leading to an optimized design of such a subsystem. Since many of the mechanization (rather than computation) factors leading to the design are compatible with optimized input processing subsystem design, the development leads to a full scale fully integrated computer system actually much more efficient than either of the subsystems comprising it. Analyses of this chapter have the goal of developing internal computation mechanizations of high duty factor i. e., the percent of time of full use of basic arithmetic capability, as one part of the full scale computer development.

6.1 THE HISTORICAL AND FUTURE ROLE OF DIVISION ALGORITHM DDA IN AIRBORNE AND AEROSPACE APPLICATIONS - This report documents the many factors implying that a DDA have division algorithm for full aerospace mission. To see the historical role of the algorithm selection rational consider the airborne applications for low speed aircraft. The conventional DDA for real time computation does not have division algorithm despite the fact that the design technique**has been known since 1954 for binary DDA with division algorithm. A partial explanation derives from the fact that the majority of airborne DDA computers are designed for conventional airborne pure inertial navigation of low speed craft with limited altitude range. The

*See Computation Capability Analysis (Chapter X); and mechanization analyses applying to computer applications requiring parallel processing capability.

**S. Cray, Remington Rand

only occurrence of a computation formally involving division in the pertinent equations involves division by radial distance from earth center. For limited altitude range of the craft the division may be avoided by a numerical approximation* of acceptable accuracy for typical pure inertial navigation applications of the past. Now that doppler radar has been developed to enable high accuracy long term navigation for low speed craft it should be pointed out that lack of division algorithm in DDA for conventional airborne navigation will be recognized as unfortunate since in the damping a division calculation is involved in transformation of doppler velocities from craft to inertial coordinates which involves relatively high frequency variables and in consequence makes division by integration or servo processes unacceptably inaccurate. **Before discussing the applications other than low velocity craft navigation which quite clearly require division algorithm, consider what the hardware and performance trade off is for conventional airborne navigation with conventional and division algorithm (where division is not formally required). Division algorithm and single increment communication imply with drum memory, in the most elementary case, one additional channel (say from four or five to five or six channels) for storage of the divisor (in which case digital processing unit iteration rate equals word rate). Also an additional transfer unit and modest amount of overflow logic must be used; communication lines must be elaborated slightly. While little increase in computer size results in the drum memory case the logical complexity is somewhat increased. Consider now the value (which we point out exists) of division algorithm in a computer used for the hypothesized application in which division is not required at all.

* Not good for aerospace missiles

** See doppler damping program and computation analysis studies

One of the more frequent uses* of division algorithm in any application is that of whole word scaling which occurs in programming 25 or 30 percent as often as general integration. Since division algorithm provides parallel generation of general integration and whole word scaling, an average increase in computation capacity of 25 to 30 percent results in applications where division is not formally required. One reason this reasonably priced modest increase in performance has not been chosen where division is not required is probably the trend toward ternary communication and computation (with twice binary precision). No ternary division algorithm was known** until developed in this contract study. There is also the consideration that a parallel (two integrations per word time) DDA with or without division algorithm requires only one additional channel over the elaborated DDA for R register in the drum memory case to double computation capacity where no division is required. Taking into account that doppler damping inertial navigation requires division the modest increase in complexity in a DDA with ternary division algorithm is seen to offer a substantial auxiliary increase in computation capacity apart from the single really essential division operation. It is therefore concluded that the long term navigator using doppler damping for low speed craft should in future systems have a DDA with ternary division algorithm (or, as will be seen, have digital Stieltjes algorithm developed in Chapter IX).

Aerospace applications very definitely require division algorithm for inertial navigation at near orbital velocities because the variation of radial distance from earth center which is prominent as the divisor in the navigation equations is substantial and has relatively rapid change. A host of other computations

* The operation Kdx is accomplished by/division by $\frac{dx}{2^C K^{-1}}$

**To our knowledge

such as orbital coordinate to earth coordinate computations and re-entry computations definitely require precision division algorithm.

6.2 HISTORICAL DEVELOPMENTS IN BINARY DIVISION ALGORITHM DESIGN TECHNIQUES AND THE PURPOSE FOR THE DEVELOPMENT OF A TERNARY DIVISION ALGORITHM COMPUTER - In a previous incremental computer design, a basic processing unit or generalized DDA "integrator" was devised which could binary increment a reciprocal in one-word time in a computer, with binary communication. There is an equivalent of the R register of the conventional integrator in the modified basic processing unit. One major difference between this previous unit and the one developed in this study is in the nature of the R register. Whereas the former employed a double length R register which had to be stored in rapid access memory at a considerable increase in machine complexity, the presently conceived device has a single word R register as does the conventional R register of an integrator which may be stored cheaply along with other register quantities on a drum. The binary computation feature must also be considered a limitation since (though binary is the cheapest in mechanization) computations in binary are subject to the "phase" effect associated with the representation of zero with a stream of alternating +1 and -1 values; also the resolution obtainable in binary computation is one-half that obtained in ternary computation. For these reasons the trend in conventional airborne computers without quotient algorithm is toward ternary design. Because quotient computation is basically more sensitive to roundoff error than other incremental computations, though less so in a quotient algorithm machine than a conventional DDA, it is expected

eg. Cray, Remington Rand. Contract No. AF 33 (038-23287).

that a ternary quotient algorithm computer would have much higher accuracy than the binary quotient algorithm computer. A binary stream of information, lacking precise representation of zero does not appear to generalize directly to a whole word binary number (despite the nomenclature) as indicated by adding bits to the short word communicated. Thus the development of multi-increment quotient algorithm for DDA type computers for higher levels of accuracy and rate handling capability does not appear to be a direct step from the binary quotient algorithm. A novel development in rate handling capability but not in accuracy was however demonstrated in an incremental computer which utilized a "variable" increment. Here a several bit word communicated represents a binary number of magnitude 2^{-K} where K, integral, is defined by the word. Clearly a very wide range of rates can be handled for a sufficiently large range of K. However, the accuracy of representation of any rate is nevertheless that of binary single increment. Basic arithmetic accuracy can be obtained only by pure multi-increment or a combination of multi-increment and variable increment. Effort in this study was therefore made in the direction of developing ternary quotient algorithm since the further development of multi-increment quotient computation appeared to be a generalization which later analyses could fully exploit.

6.3 DEVELOPMENT OF TERNARY QUOTIENT ALGORITHM

A. General Incremental Computation of Quotient Without Explicit Division Operations - The arithmetic unit of a DDA is capable of

transfer operations (single or multi-increment multiplications) but not of direct division. The quotient algorithm must consist of an incremental computation directly involving only addition, subtraction and multiplication. While approximate numerical incremental quotient algorithms have been known since 1954, it is believed that the first algorithms generally good to second order accuracy are those derived in the chapters presenting developments in the general theory of incremental computation. Since the analytical developments there hypothesize the form (to within algorithm refinements for high accuracy) of the arithmetic process used in earlier mechanizations, it may be informative to present a more brief heuristic (and less general) discussion of a quotient process. The general features of R register inputs will be delineated. Consider the reciprocal computation

$$\theta = 1/I \quad (\text{VI-1})$$

by incremental processes. Since

$$d\theta = -dI/I^2 \quad (\text{VI-2})$$

and division is to be avoided, the alternative form

$$d\theta = -\theta \cdot \theta dI \quad (\text{VI-3})$$

is one direct incremental relation capable of DDA computation provided two integrators are used. Double integration incrementation requires essentially two word times one for each serial summation, whether by ordinary DDA integrators or by a single elaborated unit with two summation registers and a single R register for output generation. Such a quotient algorithm unit of this direct type:

1. Requires two word times per incrementation.
2. Involves a single R register rather than two as in an ordinary DDA, and should obtain some reduction in roundoff error effects.

Consider now an approach which requires one word time per incrementation. The alternative difference form

$$O_n = I_n \Delta \theta_n + \theta_{n-1} \Delta I_n \quad (\text{VI-4})$$

is exact for $\theta_n = 1/I_n$.

Since the object is to compute ΔO_n^* , and unless the equation were solved for $\Delta \theta_n$ which involves division by I_n , a preknowledge of the answer is needed to satisfy the equation. Before resolving this apparent difficulty consider the practical method of holding a quantity to a minimum absolute value over periods of time. The classical approach of residue retention in this case would apparently imply computation of an R (register) quantity given by

$$R_n = R_{n-1} + I_n \Delta \theta_n + \theta_{n-1} \Delta I_n. \quad (\text{VI-5})$$

The choice of $\Delta \theta_n$ which holds R_n to a minimum absolute value is the estimate of the reciprocal increment used. Consider the case of binary representation of $\Delta \theta_n$. Since only two possible values of $\Delta \theta_n$ are acceptable for computation the problem of holding R_n to minimum value under these conditions can be approached by assuming in two separate calculations of R_n that $\Delta \theta_n = +1$ and $\Delta \theta_n = -1$. The lesser R_n which results indicates which $\Delta \theta_n$ is the preferable choice. In an ordinary DDA the R register value is pertinent, after overflow, in judging the roundoff error. Since

*where O_n is the nth output from the R register. In this particular case, $O_n = \Delta \theta_n$

nothing happens from the time after overflow to the next iteration incrementation the value of R just prior to the next incremental operation may be considered the pertinent one. In the division process if no effort were made to get the R register in the ideal state during the iteration considered, but rather to simply determine the best choice of ΔO_n which is made to be the overflow, then the adjustment to get the R register into the proper state can be made just prior to incrementation without palpable effect in the results computed. Note, that for R_{n-1} (corrected as explained above) the R register would contain (introducing a pseudo variable R_n^*):

$$R_n^* = R_{n-1} + \theta_{n-1} \Delta I_n \quad (\text{VI-6})$$

provided $I_n, \Delta O_n$ were not added. If $R_n^* > 0$ and $I_n > 0$ then, certainly, $\Delta O_n = -1$ would make R_n smaller in absolute value than $\Delta O_n = +1$. In general $\Delta O_n = -\text{sg } R_n^* \text{ sg } I_n$ is the proper choice and may be formed with simple logic using R_n^* and I_n . Having the correct ΔO_n the R register can be corrected for the term $I_n \Delta O_n$ just prior to the next iteration to obtain (with relabeling so as to consider the next iteration as the nth)

$$R_{n-1} = R_n^* + I_{n-1} \Delta \theta_{n-1} \quad (\text{VI-7})$$

The final arithmetic operation (actually carried out at the same time) is the addition of $O_{n-1} \Delta I_n$, hence the actual (rather than minimal residual) is

$$R_n^* = R_{n-1}^* + \theta_{n-1} \cdot \Delta I_n + I_{n-1} \cdot \Delta \theta_{n-1} \quad (\text{VI-8})$$

* Where I_n is the algorithm form of the divisor term in the formal residue incrementation formula.

where ΔO_n selection as $\Delta O_n = -sg R_n^* \cdot sg I_{n-1}$ insures that the effective residue of the residue retention method is minimal despite the fact that a pseudo residue actually appears in the R register. The actual incremental computation of the R register value stated above does not involve the output ΔO_n , but only the previous output resolving the "horse before the cart" implications described earlier in this description. Economical mechanizations without rapid access memory for R registers are obtainable because of this fact. The basis for precise numerical incremental formulae for more general computations involving division are developed in the chapter on general numerical incremental computation. When variables in the computation process are fed back, the formal relations stated imply the knowledge of outputs beforehand in the same way as in the above analysis. The formulae must be converted to practicable form in the same manner as in the example. The formulae derived there formally apply to whole numbers but may be applied to any given approximate number representation with the usual type of roundoff error introductions which in principle can be reduced somewhat in effect by special processing alterations.

6.4 TERNARY QUOTIENT ALGORITHM - The principle difference in binary and ternary quotient algorithm lies in the overflow criterion. The preceding description shows that the essence of overflow criterion is the selection of that one output of the set of allowed outputs for the communication type which minimizes the formal residue (rather than the actual value in the R register) associated with the principle of residue retention. Ternary communication allows three outputs +1, -1, 0. The value zero is the best choice when the

*Where I_n is the algorithm form of the divisor term in the formal residue incrementation formula.

absolute value of the R register is less than $|\frac{\tilde{I}_n \Delta O_n}{2}| = |\frac{\tilde{I}_n}{2}|$ since* in this case an output of absolute value unity would leave the formal residue greater in absolute value than before. Thus the ternary overflow criterion may be stated analytically in somewhat more general form as

$$\Delta O_n = sg R_n^* sg \tilde{I}_n \mu (2|R_n^*| > K|\tilde{I}_n|) \quad (VI-9)$$

where $K = 2^k$, k integral, and $\mu(f) = 1$ for f true, $\mu(f) = 0$ for f false.

The special device, the quotient differential processing unit (QDPU) with ternary output, introduces a decision process markedly different from that of natural overflow of an R register, yet one capable of realization. Several test calculations were performed by hand for short runs. In each case the result was correct at each iteration to within the resolution of the registers (1/2 bit), and the R register equivalent had RMS deviation from null consistent with that estimated using the roundoff theory of a normal R register. From the analytical standpoint there is no process in the unit's action which appears to imply more roundoff error than in normal R register action (on the contrary in the sense stated below).

The problem of evaluating the implications of the quotient differential processing unit (QDPU) type design in aerospace computers has been investigated in several stages with resulting generalization of the QDPU. The ultimate value of a unit capable of incrementing a wider variety of functions or functionals lies in the benefits of:

A. Alternative increased iteration rate and/or computer computation capacity.

*Where \tilde{I}_n is the algorithm form of the divisor term in the formal residue incrementation formula.

- B. Roundoff error reduction through reduction of the number of R register error sources involved in any sub-routine (as well as the increased resolution and decreased algorithm error implied by (A) for any given application).

The generalized QDPU overflow rule has several interesting properties which give insight into the nature of roundoff error associated with the R register for binary and ternary action. Regarding the contents of the R register (which is the uncommunicated residue of desired output) as an error, the conventional roundoff analysis implies that overall performance in time for a broad ensemble of computations is measured by the range of the possible R register values. The generalized QDPU overflow rule has an overflow parameter which with different selections, produces overflow of disparate types including binary, ternary, blends thereof and an odd variation. From an arithmetic operation standpoint, the overflow of R register from conceptual pre-overflow state is chosen to have the sign of the arithmetic value represented (except for a division action where the sign is reversed by a negative divisor), the difference between binary and ternary action being that ternary calls for a zero output where doing so reduces the resultant error (relative to not doing so). From the standpoint of the generalized overflow rule, the overflow parameter determines the specific situation for zero output, (for binary, never) and in general, for any overflow types according as the overflow R register arithmetic value is less in magnitude than the overflow parameter. It may be shown that the parameter value for pure ternary implies the most narrow range of R register values; thus ternary is optimum for a statistical overflow process with single increment output. Let it be granted that the test ensemble involves the occurrence of all intermediate values between the natural extremes since such can be the case for a sufficiently small Y register quantity. For a hypothesized R register value slightly less in absolute value than the

overflow parameter (for which no overflow occurs), the extreme error evidently is at least as large as K^- (the case where output is never zero, binary where $K = 0$, being trivially included). In a host of y variable situations, including the case where y is very small, the absolute value after overflow of ± 1 is $(1 - K^+)$ for $K^+ < 1$. The overall maximum error for all K satisfies

$$\epsilon = G(K^-, 1 - K^+) \approx G(K, 1 - K) \quad (\text{VI-10})$$

where $G(x, y)$ is the greater of x and y values, since overflow during computation following the start of the R register at an intermediate value prevents leaving the range assuming $|y| < 1$. The value of K which minimizes ϵ is readily seen to be $K = 1/2$ for which $\epsilon = 1/2$. This is the ternary case (variation about the initial setting of K register at $1/2$ being $1/2$ since R extremes are 0 and 1^-). The binary case is equivalently that for $K = 0$ for which $\epsilon = 1$ showing that binary has twice the residue range of ternary, the latter being optimum for single increment overflow. The QDPU, as previously reported, has the ternary action. In the division mode, the roundoff error of the output is effectively increased relative to the actual register range in proportion to the reciprocal of the scaled divisor register quantity.

6.5 MECHANIZATION FACTORS IMPLYING PARALLEL COMPUTATION CAPABILITY IN A RECIPROCAL OR QUOTIENT ALGORITHM COMPUTER -
The mechanization relations of computers with parallel computation capability and quotient computation capability are delineated without loss of implication by considering a computer which has the most elementary capability for a division process in one word time, which may be called a reciprocal algorithm computer.

Figure 6-1 shows the minimal register array enabling single increment reciprocal computation in one word time.

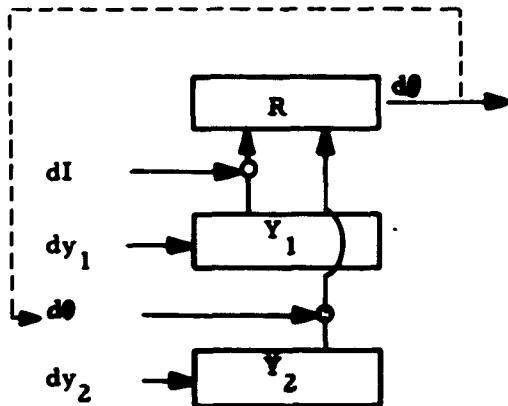


Figure 6-1. Register Array for Division in One Word Time

Two transfer operations are required for reciprocal incrementation. The cost of reciprocal algorithm is somewhat less than that of a conventional parallel DDA with two outputs indicated by the fact that there is one instead of two R registers. Consider performance of the elementary reciprocal algorithm DDA compared to the parallel DDA for several different types of computations found in important aerospace applications. Typically the number of divisions called for is small (but their accuracy has overall importance). The number of scaling operations is considerable, say 20 to 30 percent of the total program set. The quotient algorithm computer can carry out a whole word scaled integration in one word time, which as in the case of reciprocal calculation is a speed performance equal to that of the parallel DDA; of course the reciprocal algorithm enables greatly improved accuracy in reciprocal calculation. However, this effective speed performance is obtained during only about 35 percent of the program since 65 percent of some programs, in applications such as navigation, typically consist of

isolated integrations for which the reciprocal algorithm computer is no faster than one DDA integrator. During these computations the parallel DDA costing little more operates twice as efficiently. On the basis of these considerations a serial reciprocal or quotient algorithm computer is relatively inefficient in certain applications such as navigation. To obtain an efficient mechanization for a computer capable of precision reciprocal calculation the computer should also be capable of parallel (2 output) computation. Such a computer is obtained by making transfer operations programmable. Such a computer is capable of 45 to 50 percent greater speed for applications involving navigation.

6.6 INCREMENTAL CALCULATIONS ENABLED BY A THREE-TRANSFERS-TO-R-REGISTER MECHANIZATION - A basic processing unit with two transfers per iteration can be designed to do at least the work of two DDA integrators in all cases provided it is capable of two outputs. A single output unit capable of reciprocal calculation or a whole word scaled integration as a result of a computation routine structure in applications such as navigation is found to be necessarily allocated a large portion of simple integrations for which the DDA integrator equivalent is only one. The two output device involves some extra mechanization costs associated with another R-register, and the requirement of at least three inputs, and serial - parallel programmable mode capability. Three or four transfers per iteration unit with higher integrator equivalents have these requirements and not substantially more, apart from increased arithmetic complexity. Quotient incrementation of $Q = U/V$ based on the relation

$$dQ = \frac{du - Qdv}{v} \quad (\text{VI-11})$$

involves design for three transfers per iteration. Consider the more general computation

$$dQ = \frac{pdA + qdB}{v} \quad (VI-12)$$

where p , q , v are whole word variables of the form $(a_0 + a_1 A + a_2 B + a_3 p + a_4 q + a_5 v)$ and a_n ($n \geq 1$) and are expressible in the form $2^{-K} n$. An ordinary DDA requires more than 4 word times to execute (VI-12). The form of (VI-12) includes the most prevalent semi-complicated computation forms making up such computation applications as inertial navigation, atmospheric re-entry, and missile guidance. Vector computations involving scaling or common divisor often involve this form. A two transfer mechanization requires two QDPU to perform the calculation. The three or four transfer mechanization has versatility in function generation which is very impressive for the one QDPU case. The first class of computations requiring only one QDPU is of the form

$$Q = \frac{a_0 + a_1 x + a_2 y + a_3 xy + a_4 x^2 + a_5 y^2}{a_0 + a_4 x + a_5 y} \quad (VI-13)$$

Where a_n are whole word quantities, a_n are of the form $2^{-K} n$. Thus, for various choices of constants QDPU can generate x/y , $(x^{-1} + y^{-1})^{-1}$ ^{*}. A second class of computation is

$$\theta = \sqrt{a_0 + a_1 x + a_2 y + a_3 xy + a_4 x^2 + a_5 y^2} \quad (VI-14)$$

For various choices of constants the QDPU can generate \sqrt{x} , \sqrt{xy} , $\sqrt{x^2 + y^2}$, the last being useful for polar coordinate transformation. A third class of computation is the solution of a quadratic (the root being determined by initial conditions),

$$0 = \theta^2 + \theta (a_0 + a_1 x + a_2 y) + [a_1 + a_3 x + a_4 y + a_5 xy + a_6 x^2 + a_7 y^2] \quad (VI-15)$$

*The latter function used heavily in statistics for variance updating, has potential applications in adaptive control.

Adaptive missile control system studies of missile atmosphere re-entry problems relating to skin temperature involve quadratic solution computation. A fourth class of computation is x^{-K} where $K = 2^{-k}$. Related to this capability is that of generating a cubic function with whole word coefficients (output scale factor, not whole word), such as the aerodynamic fit functions used in craft control in re-entry. The inverse capability of solving a cubic is possible for a certain class with one variable coefficient. Any of these operations can be updated in one word time.

6.7 FURTHER DIGITAL PROCESSING UNIT FUNCTIONAL STUDIES BASED ON PROPERTIES OF ELEMENTARY COMPUTATIONS PREVALENT IN AIRBORNE AND AEROSPACE APPLICATIONS - Preliminary investigations of QDPU* capabilities hypothesizing alternative closely related designs were carried out in order to deduce heuristically a notion of the natural or basic computation unit implied as a result of application computation structures (as deduced from application surveys) and relative complexities of a range of mechanization structures.

A possible definition for a basic operation to be executed by the QDPU, is any operation within the broad set of computation applications in sufficient abundance that a real total saving in computation time and gain in accuracy would result, if that operation could be consistently executed in one word time (or effectively one-half word time) by hardware of acceptable cost capable of likewise doing so for all other basic operations. The key to efficient design is a generalized basic processing unit capable of executing

*Quotient Differential Processing Unit.

relatively complicated operations in one word time,* be able to execute the major portion of its operations, which are the simplest operations (principally integration or next in simplicity, whole word scaled integration), at more than conventional word time rates. The unit of more than minimal complexity and more than minimal computation capability must have parallel independent integrations capability to be efficient** in terms of duty factor. Accepting parallel mode essentiality in the QDPU, a further and very important basic operation of parallel processing type was shown to exist as determined in a survey of aerospace application computations. One of the most frequent computations in aerospace programs is vector resolution and rotation in two and three dimensions. A basic operation associated with these calculations is the two dimensional resolution. Given a variable A , compute the projection of A on axis x and y , given C_x , C_y direction cosines, namely,

$$\begin{aligned} A_x &= A C_x \\ A_y &= A C_y \end{aligned} \quad (\text{VI-16})$$

A serial DDA requires four integrators to update A_x , A_y . Note that only three whole word variables are involved just as in the other QDPU basic modes previously reported. The QDPU capable of executing the increments in parallel

$$\begin{aligned} dA_x &= C_x dA + AdC_x \\ dA_y &= C_y dA + AdC_y \end{aligned} \quad (\text{VI-17})$$

* Here operating rate is alluded to. Actually, precision level increases resulting from reduced roundoff error in themselves justify design of unit with more complex basic operation associated with only one R register (round off error source).

** See Chapter III

in one word time is not significantly more complex than the less versatile QDPU with the same input capability and register count but only three transfer units. To see the application of the generalized QDPU in three dimensional problems, consider first the polar to rectangular coordinate calculation

$$x = r \cos \theta \cos x$$

$$y = r \cos \theta \sin x$$

$$z = r \sin \theta$$

VI-18)

given $\sin \theta$, $\cos \theta$, $\sin x$, $\cos x$ and r . In this example, the generalized QDPU updates x , y , z with 100 percent efficiency as indicated in Figure 6-2.

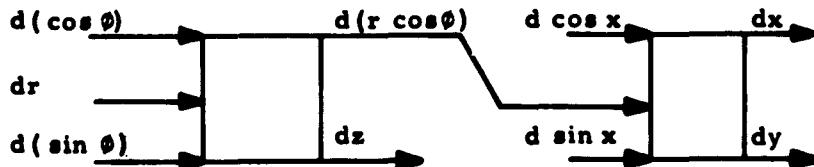


Figure 6-2. QDPU Program Diagram for Updating x , y , and z with 100 Percent Efficiency

This mechanization accomplishes in two word times the computations for which previous mechanizations required from 4 to 8 word times. An example of the proposed QDDA speed performance, relative to other existing DDA computers, is a typical program for a toss bombing problem chosen to illustrate the capability of an earlier machine. The example is especially favorable to the earlier machine because few isolated integrations are called for as in the more important case of navigation computations for which the QDDA however maintains the same efficiency (see Chapter X). The QDDA program is diagrammed in Figure 6-3. Two more conventional serial DDA performances are summarized for the following computation subroutine:

A PORTION OF THE PROGRAM DIAGRAM
FOR THE TOSS BOMBING PROBLEM

$$\begin{aligned}\sin \lambda &= \frac{\frac{1}{2} g \tau^2 + H \cos(\lambda - \sum) - w_r (\sin(\lambda - \sum))}{D} \\ \tau &= \frac{D \cos \lambda + (\frac{1}{2} g \tau^2 + H) \sin(\lambda - \sum)}{V + w_r \cos(\lambda - \sum)}\end{aligned}\quad (VI-19)$$

INPUTS: V, w_r, \sum, D, H

CALCULATE: $\sin \lambda$

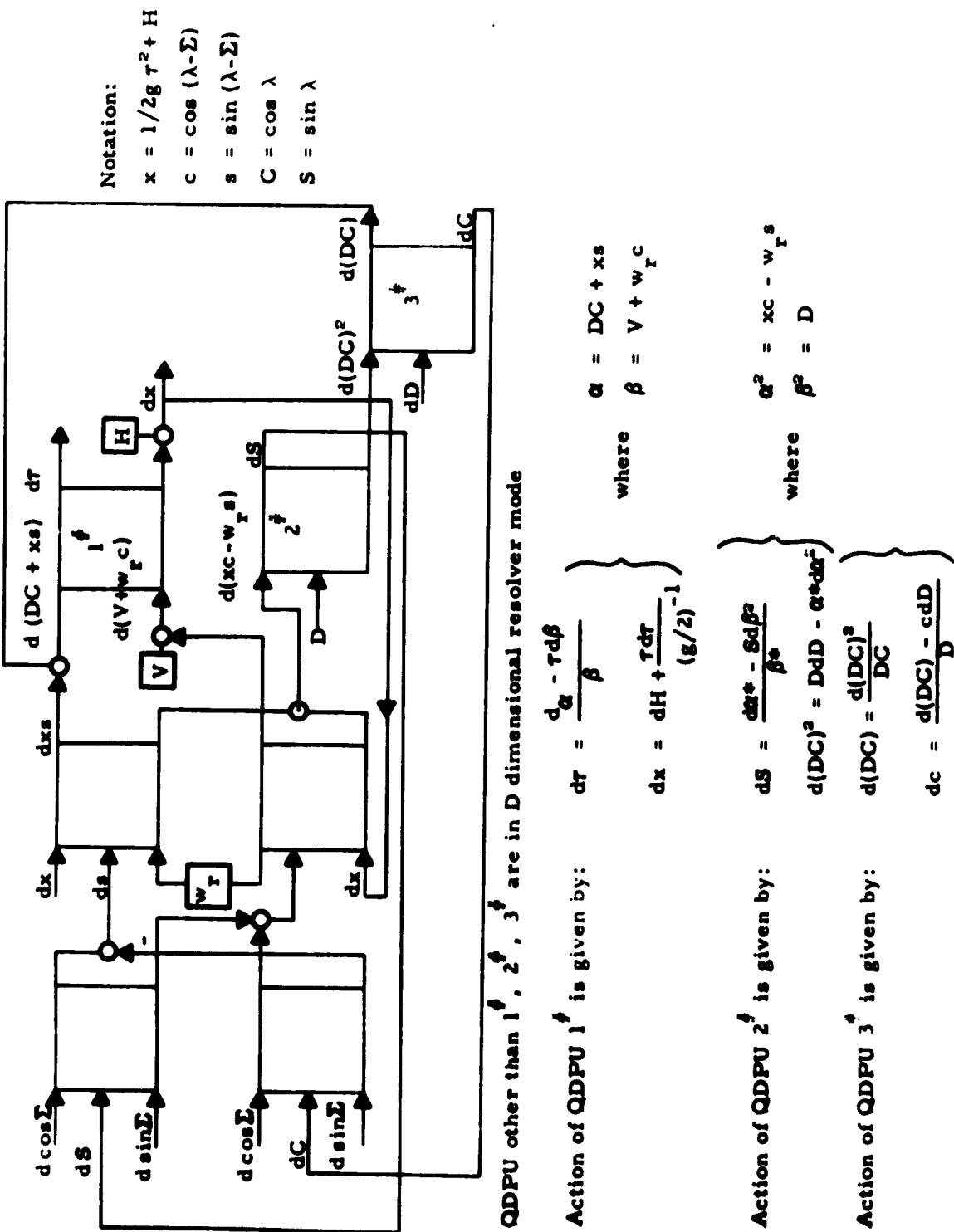


Figure 6-3. QDDA Program Diagram For Toss Bombing

<u>SPEED PERFORMANCE</u>	<u>SPEED FACTOR</u>
QDDA - 7 words/iteration (diagram)- (Inertial QDDA Operation Structure)	200%
Variable Increment DOA with Quotient Algorithm 14 words/iteration	100%
Conventional Serial DDA - 29 words/iteration (estimated)	48%

(This does not reflect equivalent speed increase due to the unique QDDA higher order ternary algorithm, but only hardware iteration rate.)

Relative accuracy performance is effected by the iteration rate as well as integration algorithm. The later developed QDDA multi-increment computation and also higher order integration algorithm of QDPU described in the chapter on QDDA algorithm and round-off operation raises equivalent iteration rate to an additional level higher than indicated above for existing incremental computers. In terms of hardware iteration rate alone it is noted in the example program that the QDDA with stated operation features presents almost the same step over the variable increment DOA that the latter computer presents over conventional serial DDA. In navigation computation the step forward is yet greater. It is important to evaluate the relation of an one example program to the larger set of application programs, especially for the aerospace case explored in considerable detail in Chapter XI. As previously reported, certain important applications, most notably inertial navigation, imply somewhat different relative performances of the various computer types as a result of more frequent occurrence in their routines of certain basic operations. In the extreme case (no pure example of which is believed likely to be found*) where pure non-additive integrations in the routine

*Nor at all in the ODDA as seen later, since input processing may use a double precision mode with 100% duty factor.

predominate over all others, the variable increment DDA because of a lack of parallel computation ability, would perform only 20 percent faster (rather than 108 percent faster, as in the previous example) than a serial DDA, the 20 percent gain being primarily ascribed to its whole-word scaling capability. In this extreme (conceptual) case the QDDA operation as stated in this first analysis would perform 140 percent faster (in later analysis of multi-increment QDDA the performance is 180 percent faster* for this extreme conceptual case) than a serial DDA (instead of 316 percent faster as in the previous example) again considering iteration rate alone. At the other end of the spectrum of application computations in which coordinate, vector, and tensor transformation routines predominate, the QDDA, because of the resolver mode, has four times the speed of a serial DDA (and still about twice the speed of the variable increment DDA.) On these bases, the toss bombing program described above, is considered to give a fairly accurate (or slightly conservative) indication of QDDA pure program speed performance on the basis of stated operation features relative to previously designed variable increment DDA performance.

It is also important to gauge performances relative to serial DDA, especially for the inertial navigation application. An airborne navigation system developed at Litton Industries and designed for minimum overall complexity was chosen for the preliminary evaluation (the program length is probably only 25 percent of the length required for a full aerospace mission). A QDDA program was derived for minimal complexity inertial navigation which involves only 16 QDPU and (with the conservative 200 kc clock rate) could achieve an iteration rate of as high as 625 iter/sec. A serial DDA requires 54 integrators and at best, for this clock rate, would have an iteration rate of 200 iter/sec. Had the DDA been allocated such calculations as arc sin, etc., instead of the GP, the comparison would be much improved

*But with programmable double precision.

for QDDA. The general results of these preliminary studies were considered sufficiently encouraging to make more detailed studies in the direction of developing a computer for a full aerospace mission.

6.8 FURTHER STUDIES OF PROGRAMMABLE TRANSFER FEATURES AND REGISTER REQUIREMENTS FOR A BASIC DIFFERENTIAL PROCESSING UNIT CAPABLE OF DIVISION WHICH HAS MAXIMUM DUTY FACTOR - Four major factors imply the DDA design approach of developing a differential processing unit consisting of a combined integrator ensemble with serial-parallel processing features:

1. Precision division computation is required for aerospace applications (for missiles of near orbital velocity and wide altitude range), implying several transfer actions within the basic unit cycle.
2. Parallel transfer capability without parallel output capabilities, the latter costing little more, limits processing rate for applications such as navigation.
3. Demand for a high computation capability, as a result of the large aerospace computation programs and high precision requirements.
4. The full scale computer system with simplest mechanization that is capable of input processing and internal computation functions has a single arithmetic module which is time shared for these functions. The minimum level of complexity for the input processing implies overall low cost mechanization of the combined integrator ensemble with serial parallel processing features of complexity up to that required for input processing.

One useful measure of the efficiency of a differential processing unit is the duty factor i.e., percent of full time employment, during execution of a computation program, of available arithmetic capability toward a useful

end. It will be assumed that all mechanization features which do not limit processing versatility or duty factor are optimized. The appropriateness of a differential processing unit, with a high duty factor in a DDA system tailored to the application, is the degree of match between the computation capability available and that demanded by the application, since realizing a high duty factor as the assumed design goal implies minimum mechanization cost for the computation capacity obtained. It was decided that quotient algorithm is required for the aerospace application, and, associated with these requirements, parallel computation design for high duty factor in the broad set of computations. A quotient differential processing unit, QDPU, should be capable of two outputs and probably not more. Restricting consideration to two output devices, the problem of attaining high duty factor with the simplest mechanization involves investigation of such features as the number of input variables (each input variable being the sum of several outputs of a QDPU or external input) involved in transfer operations and, in addition, the number of y registers. The number of transfer units determines an upper limit on computation capacity which must match the application. Studies in single increment QDPU's consider two to four transfer designs, two being the minimum for precision reciprocal computation. The number of input variables involved in transfer operations necessary to enable high duty factor depends on the nature of program computations. Airborne and aerospace calculations typically involve vector and coordinate transformations, which are composed of elementary operations, pairs of triples of which have a common variable. It will be shown that provisions for three input variables, each of which may be the sum of several outputs, is generally necessary and adequate to exploit the arithmetic capability of a QDPU designed to execute up to four transfer operations. There is minor mechanization cost in providing for selecting the communication of four inputs (assuming the same total of component inputs to the QDPU) in the case of rapid

access stored outputs, but the additional cost for component variables summation, quantization, storage for residue retention, and the programmable selection for transfer operation, are significant. The fact that three input variables enable high duty factor for a four transfer QDPU, as well as for a two transfer QDPU, is favorable for the more powerful associated computer. Another closely related consequence of the type of computations for airborne and aerospace applications is that both three and four transfer QDPU's can have high duty factor with a mechanization with three y registers. Of course, the two transfer QDPU must have two y registers for reciprocal calculation, and cannot use more than two y registers with two transfer capability. Component inputs forming an input variable must be quantized with residue retention registers for good accuracy. These registers are referred to as δ registers. For single increment computation they must be four to seven bits in length for flexible scaling, whereas for multi-increment the computation must be seven to ten bits in length. Therefore, they are generally cheaper in storage costs than y and R registers unless stored in separate drum channels, one to a word time. In this case considerable storage space is either wasted or available for other purposes. The extra space can be used for inputs address, as in the drum storage case the δ registers do not basically have expensive storage. Multi-input quantization operations in generating δ registers residues and quantized variables can be mechanized economically using time shared arithmetic subunits.

6.9 MECHANIZATION SIMPLIFICATIONS RELATIVE CONVENTIONAL PARALLEL DDA WHICH EFFECT TIME SHARING WITHOUT RATE LOSS IN MULTI-REGISTER SYSTEMS WITH MANY TRANSFERS PER WORD TIME - The increased computation capacity of systems with many transfers per word time has been considered by many to be proportionately paid for by increased mechanization requirements. This concept is reflected in the eventual integration of internal computation with input processing the latter

alone requiring a set complexity level. It should also be independently concluded that a DDA with many transfers per word time without time shared input processing may have surprisingly efficient mechanization by incorporating certain time sharing features which do not imply iteration rate loss. The element of mechanization cost in proportion to the number of transfers effected per word time basically required by application and bit rate is only one part of overall cost, and certainly if it were the only added cost for several transfer capability (single bit), would make the overall cost of the more powerful computer only fractionally somewhat more costly than a conventional programmable DDA. The particular mechanism chosen for communication to the QDPU is simpler than conventional DDA, of large program capacity because output number is reduced by one half in the QDPU relative to conventional DDA.

The costs of providing input selection of accessible outputs and multi-input quantization per word time mechanization with modal versatility would appear on first inspection to imply proportionate cost with computation capability. Closer analysis reveals that these costs are not nearly of such degree, provided there is full exploitation in design of basic input operation characteristics. Adequate scaling flexibility is provided by multi-input registers which are less than one-half to one-fourth the length of the Y and R registers. This implies that a single arithmetic mechanism for the multi-input quantization operation can, by time sharing, produce quantized inputs of two to four in number, thereby costing little more in flip-flops than in the single transfer DDA, provided in the case a few words of rather cheap core storage is available. A significant saving results in channel count using drum storage results.

Consider the y and Δx allocation for several-input, several-transfer mechanizations with a high level of transfer versatility obtained by programmable mode design. Programmable Δx allocation to transfer units for y variables is effected by a program bit storage in flip-flops during the word time in the four transfer per word time QDDA. The allocation would appear at first count to require eight flip-flops for arithmetic operation, ten flip-flops for channel storage, and up to ten flip-flops for transfer modality. However, using a time shared multi-input quantization register and only two channels for Δx data storage, the inputs designated by a seven bit address are selected from a core memory, the quantization register contents after computation completion, (1/2 to 1/4 word time), may be put, after word split, to form residue which is restored and transfer bit (or bits) used in arithmetic operation of that QDPU. It is estimated that application of this design technique for a four transfer unit reduces flip-flop requirements relative to the proportionate hardware approach from 28 flip-flops to five flip-flops and 20 cores (the latter being relatively cheap).

6.10 HEURISTIC COMMENTARY ON COMPUTER DESIGN - As stated by Claude Shannon* in a heuristic analysis of desirable computer attributes, the human brain is a suitable computing system supplied by nature that, in our state of advancement in the computing field, offers much for enlightenment in computer design. While most facts are unknown regarding brain operation, he states it is clear that the brain has extensive parallel operation features. In reflecting on this advice and statement of fact, the comments of other authorities in the field of psychology are believed pertinent. While undoubtedly neuron action has mega-parallelity, the flows of what

*Litton Consultant

may be said to constitute individual thoughts may not be in such degree. Indeed, psychologists state that the healthy brain can carry no more than three simultaneous continuous thought processes, and most healthy brains (of equally intelligent individuals) can carry only two. It is perhaps a surprising coincidence that the basic operations, which most frequently occur in computations, have a level of complexity and interoperation correlation that leads to greatest efficiency in mechanization with a comparable degree of parallelity found in human thought processes. The QDDA mechanization developed exhibits a close parallelism to the human brain. Each operation element (one word time of computation) possesses the ability to carry on two simple and parallel operations simultaneously or, alternatively, to execute a single complex operation, thus utilizing its computation capabilities maximally at all times.

CHAPTER VII

NEW CONCEPTS OF MULTI-INCREMENT COMPUTATION AND DEVELOPMENT OF A MULTI-INCREMENT GENERAL (QUOTIENT) ALGORITHM COMPUTER WITH SECOND DIFFERENCE OUTPUTS HAVING SIMPLIFIED COMMUNICATION

7.0 INTRODUCTION - The need for multi-increment computation* in certain portions of airborne and aerospace computation programs has been established in extensive studies during the program. In Phase I the concept of using different computation techniques on routines with markedly different computation requirements was given form in the classification of input processing and internal computation types. In Phase II it was established that aerospace computations require division capability which presents another type of computation requirement. Internal computations were classified as those with the more readily met computation requirements, which by the nature of application programs tend to constitute the major portion of program operations count. The goal of developing a full scale programmable computer capable of executing all types of computations, a computer which has moderate mechanization requirements, was seen to imply the design approach of modal operation with extensive time sharing of arithmetic and communication hardware. Time sharing implies some price in program capacity for the internal computations in particular. Therefore an increased computation sophistication is implied for internal computations in a computer using time sharing (for mechanization simplicity) relative to one without time sharing. Since input processing requires multi-increment computation implying the presence in the computer of a many bit transfer mechanism, the possible best selection of several bit increment computation for internal computation was considered eminent. The integration of

*Or single increment computation at ultra high iteration rate which implies corresponding hardware costs with state of the art hardware.

input processing into the computer system in an efficient manner also implied that data storage and communication channels be similar or analogous to that of a conventional DDA. Therefore investigations were launched in the field of multi-increment DDA design in the direction of several bit increment computation (as well as the whole word increment computation demonstrated in the strap-down processor). Precision quotient algorithm was considered essential for the final computer developed.

7.1 LIMITATIONS OF STATE OF THE ART COMPUTER DESIGN TECHNIQUES IN MULTI-INCREMENT, VARIABLE INCREMENT AND QUOTIENT COMPUTATION -

- A. General Design Factors -** The historical motivation for single or few bit rather than many bit increment computer design clearly stems from the desire for simplified computer mechanization associated with costs for communication and multi-transfer (multiplication). Generally the less the increment size the less the attainable precision so that the application determines the tolerable minimum increment bit length. It will be shown later in this chapter that the general impression regarding inherently increased communication costs in a multi-increment computer is incorrect.
- B. Variable Increment Computation Limitations -** Variable increment computation techniques, employ single transfer and call for, at any time during operation, effective communication of a single binary increment of variable scale; the scale of the output is 2^K , K integral is communicated as a several bit word which changes as the required output rate dictates. A single transfer computer in Stieltjes integration operations or product computations cannot approach the accuracy of a multi-transfer computer whether variable or constant scale

communication is used or not. In variable increment computation, when output rates lay in a general range of magnitude, the communications are single bit magnitude with a constant scale, and can have no greater accuracy than a single increment computer during that period. The overall performance of a variable increment computation is expected to be somewhat superior to single increment computation provided the periods of maximum rate change of outputs is short. The rate handling capability, in a sense, is the major asset of the variable increment computation. Multi-increment computation generally provides both rate handling and precision capability, the latter being much lower in variable increment. The communication requirements of variable increment present a significant additional cost relative to single increment DDA. Those for variable increment are comparable to the costs for direct \geq 4-bit multi-increment communication. This study shows how the superior multi-increment computation may be mechanized in a form using single increment communication which is markedly cheaper than that of existing variable increment computers.

- C. Multi-Increment Computation with Quotient Algorithm - The desirable feature of quotient algorithm appeared to present a stumbling block in the development of multi-increment computation. It is deduced that the development of variable increment was an effort to walk around rather than remove the apparent stumbling block. Apart from lack of design techniques, the major problems in multi-increment design arise from the cost of communication and multi-transfer. The development during this study of design techniques to accomplish multi-increment quotient algorithm computation, and further to hold communication costs to those comparable to single increment computation (and less

than variable increment), eliminates two problems. The cost of multi-transfer mechanization for internal computation in a computer, capable of input processing, is shown to be minimal in a time sharing design.

- D. Initial Problems in Developing Design Techniques for Multi-Increment Computation with Division Algorithm - The technical design problem of multi-increment computation with division algorithm in a computer, in which only transfer operations are executed, stems from the nature of output generation from the DDA integrator (or generalized differential processing unit). Conventional single increment DDA integrator outputs are generated by natural overflow, i. e. the propagation of a carry from the most significant bit position of the integrator. Previous analyses* in multi-increment computation without division algorithm, showed that natural overflow is not appropriate in multi-increment computation. The output should be generated by one of the alternative round off techniques. The design approach is based on the preservation of the relationship of pre- and post-overflow difference equaling output magnitude as is normally attained by natural overflow. The development of more sophisticated algorithms is expedited by analyzing output criterions and R register adjustments for outputs. The fundamental difference in single increment computation between (1) incrementing a variable and (2) incrementing a variable and dividing by a whole word variable, in an incremental computer capable only of transfer operations, may be described in terms of constant unit and

*Analysis by J. Campeau at Litton Industries in 1957.

variable whole word scale factors. The output criterion for (1) might be described abstractly as calling for an output if the subtraction or addition of unit (scale relative to full register value) would, in the case of binary or in ternary, reduce the absolute value of the number in the R register (relative to the original value for ternary). For (2), the output criterion calls for an output if the subtraction or addition of the whole word (divisor) variable (or 2^{+K} times it if desired) would reduce the absolute value of the number left in the R register; the output for (2), if non-zero, is ± 1 in scale chosen for the output which may be chosen 2^K times that of another choice without making the output determination in a binary computer more difficult (effected by relative delay). Variable increment computation can be obtained by mechanizing choice of scale 2^K for comparison (according to variable rate, if desired for maximum rate handling). Mechanization of output criterion for binary merely requires using the sign of the R register and the divisor. In ternary, the mechanization is the same except for a modification to have zero output if the R register contains less than half the divisor, as determined by two parallel computations of sum and difference. In principle, the output criterion for the multi-increment case can be determined by many parallel test computations for each possible alternative of output, with the number of test computations being $> (2^{M+1})$ for M bit increment. Evidently the number of adders required to perform such a direct test approach is unacceptable. An indirect design approach which leads to relatively simple mechanization is developed in a later section. The second problem, which is correcting R register for each output (since overflow is not natural), appears in the first quotient

algorithm computer in 1954 where it is deduced rapid access registers were used for correction for (a posteriori) deduced outputs. The quotient algorithms developed here are amenable to drum register storage for economy, since deduced outputs may be feedback at the next iteration to correct the R register during the next operation simultaneous with the normal operations of that cycle.

7.2 FEASIBILITY OF SINGLE INCREMENT COMMUNICATION FOR A MULTI-INCREMENT DDA COMPUTING BAND LIMITED VARIABLES - The prevailing impression exists in the computer field that a multi-increment DDA must have multi-increment communication in order to have multi-increment accuracy. Certainly it is true that the integrator or generalized basic processing unit must have the information regarding multi-increment changes in order to have the precision. The question, then, is really whether the information is available without direct multi-increment communication. A well known principle of information theory is that information transmission rate may be low for band limited variables as compared to that for high frequency variables. Thus, if the internal computations (which are generally lower frequency than those input processing) have sufficiently low frequency relative iteration rate, then some level of multi-increment accuracy may be possible using single increment communication of the right kind. Since DDA computations are generally considered based on high degree of analyticity (apart from interruptions of analyticity, which can often be handled using decision modes or, if necessary, GP supervision) it is expected that most variables in a DDA program are band limited. The most elementary condition for simplified communication is that the variables, represented by the communicated increments, change not more than a certain amount per iteration i.e. for properly scaled variables represented by M bit increment that do

not change by more than 2^{-M} fraction in one iteration. Certainly if M is sufficiently small the physical properties of the variable permit this assumption, just as the scaling assumed on a physical basis holds for the total increment. Consider a sinusoid of frequency f being computed at iteration interval τ with multi-increment accuracy. The increment may have the form

$$\Delta x = A \sin (2\pi f \tau n + \theta) \quad (\text{VII-1})$$

The change in Δx per iteration is

$$\Delta^2 x \sim 2\pi f \tau A \cos (2\pi f \tau n + \theta) \quad (\text{VII-2})$$

The maximum values are $\Delta x_{\max} = A$ and

$$\Delta^2 x_{\max} = 2\pi f \tau A, \quad \text{hence we have } \frac{(\Delta^2 x)_{\max}}{(\Delta x)_{\max}} = 2\pi f \tau \quad (\text{VII-3})$$

Most variables in aerospace computations are significantly less than 0.1 cps hence a computer at 150 iter/sec with single increment communication and having many bit transfer capability could be scaled to actually realize a multi-increment accuracy consistent with

$$\frac{\Delta^2 x_{\max}}{\Delta x_{\max}} \leq \frac{1}{230} \leq 2^{-7} \quad (\text{VII-4})$$

namely, 7 bit increment accuracy with single increment communication.

Suppose that 3 bit multi-transfer is mechanized, then the maximum frequency sinusoid consistent with single increment communication is

$$f_{\max} = \frac{1}{2\pi\tau} \frac{\Delta^2 x_{\max}}{\Delta x_{\max}} = \frac{1}{2\pi\tau} 2^{-3} \quad (\text{VII-5})$$

At 150 iter/sec, as for internal computation, $f_{\max} \sim 3$ cps. At 600 iter/sec for 6 bit increment computation, using 2 bit increment communication as for input processing, $f_{\max} = 6$ cps. The maximum frequencies apply to general variables, whose major component have the stated maximum frequency. For minor components of perhaps higher frequency, the reciprocal of the relative fraction of full amplitude of the component affects allowable frequency in direct proportion. The highest frequency variables, such as in air data or strapdown applications, have major components at < 0.5 cps and perhaps some minor components at relatively high frequencies. While simplified communication appears possible for input processing, the relatively small number of communications required does not require use of the simplified communication technique. Rather, the internal computations characterized by a gross number of communications, resulting from the typically large program, may properly use the simplified communication with significant saving and also generally be programmed for the full accuracy possible by a several bit transfer capability on the generally low frequency variables assigned.

7.3 ALGEBRA OF SCALING OF A MULTI-INCREMENT QDDA WITH SECOND DIFFERENCE OUTPUTS OF THE QDPU - All DDA mechanizations involve register storage and communication of information representing physical quantities (or problem quantities in a simulation computer) that are subjected to arithmetic operations of variable updating and transfer (conditional accumulation to R registers). Interpreting any full register of any fixed length, as having a machine arithmetic value of 1, a given register,

having the purpose of storing a given quantity X , has an associated physical scale $S^P_{(x)}$, which is chosen for overall most accurate computation consistent with machine operation constraints. In addition, to the quotient algorithm (developed for ternary communication) the computer evolved during the latter portion of the study generates (1) second difference outputs (rather than first difference as in all existing DDA's) (2) utilizes in each QDPU additional registers (first difference storage) for accumulation of ternary communications (single increment and sign), and (3) utilizes multi-increment updating and transfer operations (3 bit or 6 bit). The design features, (1) and (2), which are major new design features, are chosen for overflow mechanization and communication hardware simplicity in achieving (3), the latter having been widely discussed in the DDA field, but not, to our knowledge, materialized in a computer. An algebra of scaling the multi-transfer QDDA was derived not only for the basic purpose of programming application computations, but also for the exploration of the implications of general operation for optimized design. The choice of definition of machine numbers, where registers contain maximum machine values of 1, makes the y register and independent variable register closely analogous to a single increment, single transfer DDA where a fraction instead of a ± 1 , 0 is added to a y register with the same scale factor, and a product of a fraction instead of ± 1 , 0 with an integrand quantity representing the independent variable increment evolves the R register increment with the same scale factor. The more general case of a programmable scale factor for transferred quantities was analyzed to better coordinate the several transfer actions in a QDPU (only one transfer takes place in a conventional DDA). Scaling of the QDPU outputs (second differences) for accumulation in first increment registers is a function of the overflow mechanism which has been developed and the optimal system performance choice for the physical variable represented, which is consistent with the single increment communication and transfer bit length mechanized.

Analysis will use the following general symbology,

$r(x)$ is the machine number in the register associated with physical variable x where full register is considered to contain ± 1

S_x is the scale of x in the register i.e., the physical magnitude of x for which the register contains ± 1

Then in general $x = S_x \cdot r(x)$.

Analysis is assisted by attributing to contents of the R register the representation of a physical quantity R , in which case $R = S_R \cdot r(R)$ where $r(R)$ is the R register contents. Analysis of multi-increment computation with 2nd difference output where a division process is executed, may be simplified by representing the updating of R in terms of the sum of normal transfers associated with one or more integration processes ΔI_n , and the transfer associated with division (by v) in which case

$$R_n = R_{n-1} + \Delta I_n - v_{n-1} \Delta \theta_n^D \quad (\text{VII-6})$$

The quantity $\Delta \theta_n^D$ being the updated first difference of outputs after feedback as distinguished from $\Delta \theta_n$, which is the updated first difference including present output. Thus $\Delta \theta_n = \Delta \theta_n^D + \Delta^2 \theta_n$. The computation to be executed is

$$\Delta \theta_n = \frac{\Delta I_n}{v_n} \quad (\text{VII-7})$$

In terms of machine register contents and physical scales we have

$$S_R \cdot r(R_n) = S_R \cdot r(R_{n-1}) + S_{\Delta I} \cdot r(\Delta I_n) - S_v S_{\Delta 0} r(v_{n-1}) \cdot r(\Delta 0_n^D) \quad (\text{VII-8})$$

Transfers associated with integration will be executed to have the same scale as the R register, hence $S_{\Delta I} = S_R$ and

$$r(R_n) = r(R_{n-1}) + r(\Delta I_n) - K r(v_{n-1}) r(\Delta 0_n^F) \quad (\text{VII-9})$$

where

$$K = S_v S_{\Delta 0} / S_{\Delta F}$$

The quotient algorithm developed does not involve direct "overflow" but rather an "output" according to criterion with the effect in computation of overflow being produced by feedback of "output." Scaling of output, which depends on the output mechanism, is elucidated as follows: For the feature of ternary output where zero is the output (when minimum error is achieved by doing so) the latter would be appropriate (without residue retention) if

$$\left| \Delta I_n - v_{n-1} \Delta 0_{n-1} \right| < \frac{|v_{n-1} \Delta 0_n|}{2} = \frac{|v_{n-1}| |\Delta 0_n|}{2} \quad (\text{VII-10})$$

for $\Delta 0_n$ chosen non-zero. With the desirable computation feature of residue retention, output is zero if

$$R_n - \frac{|v_{n-1}| |\Delta 0_n|}{2} < 0$$

Regarding machine outputs as +1, -1, or 0 in the convention of ternary, the outputs are regarded as contained in single bit register (not including sign).

Then

$$\left| \Delta^S O_n \right| = \left| S_{\Delta^S O} \right| \left| r(\Delta^S O_n) \right| \quad (\text{VII-11})$$

where $r(\Delta^S O_n) = +1, -1, \text{ or } 0$. For $\Delta^S O$ hypothesized non-zero, for application in the output criterion inequality,

$$\left| r(\Delta^S O_n) \right| = 1$$

hence, the test inequality is

$$\left| R_n \right| - \frac{\left| V_{n-1} \right| S_{\Delta^S O_n}}{2} < 0 \quad (\text{VII-12})$$

Expressed entirely in register quantities and scales involved, the test inequality is

$$\left| r(R_n) \right| - K^* \left| r(V_{n-1}) \right| < 0 \quad (\text{VII-13})$$

where

$$K^* = \frac{S_{\Delta^S O} S_v}{S_R} = \frac{S_{\Delta^S O} S_v}{S_{\Delta I}}$$

The output criterion, in terms of register and physical scale quantities is

$$r(\Delta^S O_n) = \text{sgn}(r(R_n)) \text{sgn}(r(V_{n-1})) u \left(\left| r(R_n) \right| - K^* \left| r(V_{n-1}) \right| \right) \quad (\text{VII-14})$$

providing $S_R, S_v > 0$, and taking u to be the unit step function. Assuming $r(\Delta^S O)$ is added to $r(\Delta O)$ according to the relation

$$\Delta r(\Delta O) = 2^{-S} r(\Delta^S O) \quad (\text{VII-15})$$

multiply both sides by $S_{\Delta\theta} \cdot \frac{S_{\Delta^2\theta}}{S_{\Delta^2\theta}}$, obtain

$$\Delta(\Delta\theta) = 2^{-8} \frac{S_{\Delta\theta}}{S_{\Delta^2\theta}} \cdot \Delta^2\theta \quad (\text{VII-16})$$

hence $S_{\Delta^2\theta} = 2^{-8} S_{\Delta\theta}$ (VII-17)

Then the constants K and K^* for R register incrementation and output generation are seen to be related by

$$K^* = 2^{-8} \frac{S_v S_{\Delta\theta}}{S_{\Delta I}} = 2^{-8} K \quad (\text{VII-18})$$

Summarizing the R register incrementation and output generation equations, we have

$$\begin{aligned} r(R_n) &= r(R_{n-1}) + r(\Delta I_n) - K r(v_{n-1}) r(\Delta\theta_n) \\ r(\Delta^2\theta_n) &= \text{sgn}(r(R_n)) \text{sgn}(r(v_{n-1})) u \left(r(R_n) - 2^{-8} K |r(v_{n-1})| \right) \end{aligned} \quad (\text{VII-19})$$

where

$$K = S_v S_{\Delta\theta} / S_{\Delta I}$$

assuming $S_v > 0$, $S_{\Delta\theta} > 0$ (the latter since $S_{\Delta\theta} = S_R$).

To elucidate transfer relationships consider the case where $\Delta I_n = \tilde{y} \cdot \Delta X$ where \tilde{y} is the integration algorithm modified y of same scale and where y is scaled unity in r(y). Then

$$\Delta I_n = r(\tilde{y}) r(\Delta X) S_{\Delta X} \quad (\text{VII-20})$$

and $S_{\Delta I} = S_{\Delta X} = S$ (VII-21)

Suppose v is also stored with unit scale and $S\Delta O$ has the same scale as Δx , namely $S_{\Delta \theta} = S$. Then $K = \frac{(1) + (S\Delta x)}{S\Delta x} = 1$ and the R register incrementation equation in this case is

$$r(R_n) = r(R_{n-1}) + S \left[r(\tilde{y}) - r(\Delta x_n) - r(v_{n-1}) r(\Delta \theta_n^D) \right] \quad (\text{VII-22})$$

with output criterion

$$r(\Delta^2 \theta_n) = \text{sgn}(r(R_{n-1})) \text{sgn}(r(v_{n-1})) U(|r(R_n)| - 2^{-s} |r(v_{n-1})|) \quad (\text{VII-23})$$

The transfer operation for integration and division operations are identical except for sign in this case, for any choice of scale of $S\Delta^2 O$ provided the output criterion is adjusted so that s satisfies

$$S\Delta^2 \theta = 2^{-s} \cdot S \quad (\text{VII-24})$$

Actually, for $\Delta^2 \theta$ to have a single increment representation satisfying a choice of $S\Delta^2 \theta$ the physical variable $\Delta^2 \theta$ must be known to be rate limited to the extent that a fractional change in $\Delta \theta$ in one iteration does not exceed 2^{-s} .

7.3 GENERAL (QUOTIENT) ALGORITHM FOR THE QDD²A. BASED ON THE NUMERICAL COMPUTATION ANALYSIS - The analysis of Chapter II led to the quotient algorithm for whole word computation in which outputs $\Delta^2 \theta_n$ are second differences generated using the calculation*

$$R_n = R_{n-1} + F_n \Delta x_n - \tilde{v}_n \Delta \theta_n - \tilde{v}_{n-1} \Delta^2 \theta_n \quad (\text{VII-25})$$

where ($\tilde{}$) means a variable is algorithm modified and where the desired calculation is

$$\Delta \theta_n = \int_{(n-1)\tau}^{n\tau} \frac{pdx}{v} \quad (\text{VII-26})$$

Consider along with this computation the more general computation (readily obtained by generalizing the analysis of Chapter II),

$$R_n = R_{n-1} + \tilde{P}_n \Delta X_n + \tilde{q}_n \Delta Y_n - \tilde{V}_n \Delta \theta_n - V_{n-1} \Delta^2 \theta_n \quad (\text{VII-27})$$

where the outputs $\Delta^2 \theta_n$ are second differences and

$$\Delta \theta_n = \int_{(n-1)\tau}^{n\tau} \frac{pd़x + qd़y}{v} \quad (\text{VII-28})$$

When $v = 1$ this computation reduces to

$$R_n = R_{n-1} + \tilde{P}_n \Delta x_n + \tilde{q}_n \Delta y_n - (\Delta \theta_n + \Delta^2 \theta_n) \quad (\text{VII-29})$$

where

$$\Delta \theta_n = \int_{(n-1)\tau}^{n\tau} pd़x + qd़y \quad (\text{VII-30})$$

In mechanization for multi-increment computation the later computation requires two multi-transfer units for parallel operation (one for $\tilde{P}_n \Delta x_n$ and one for $\tilde{q}_n \Delta y_n$) and one single transfer for $(\Delta \theta_n + \Delta^2 \theta_n)$ as will be discussed later. The same multi-transfer unit requirements hold for the quotient algorithm first stated.

The application studies of Chapter XI show that the calculations

$$\Delta \theta_n = \int_{(n-1)\tau}^{n\tau} \frac{pd़x}{v} \quad (\text{VII-31})$$

$$\Delta \theta_n = \int_{(n-1)\tau}^{n\tau} pd़x + qd़y \quad (\text{VII-32})$$

include the majority of basic calculations desired in an incremental computer. The fact that multi-transfer unit requirements in parallel computation are the same for the two calculation types, implies that a basic unit which can be programmed for either operation can be efficient most of the time during the

computation cycle in executing a typical application program, since the arithmetic capability of the unit is fully used most of the time. The first stated algorithm can be put in the form

$$R_n = R_{n-1} + \tilde{p}_n \Delta x_n + \tilde{v}_n \left[-(\Delta \theta_n + \Delta^2 \theta_n) \right] + \Delta v_n \Delta^2 \theta_n \quad (\text{VII-33})$$

in which the total multi-transfer and single transfer requirements are the same as the sum of integrals algorithm. A unit which is programmable for both the operations of form

$$R_n = R_{n-1} + \tilde{p}_n \Delta x_n + \tilde{v}_n \Delta y + \left[-(\Delta \theta_n + \Delta^2 \theta_n) \right] \quad (\text{VII-34})$$

and the above has the basic computation capability discussed. The general theory calls for integration algorithm of component terms (indicated by $\tilde{}$) which depends purely on the lagged or unlagged nature of the variable, for which

$$\tilde{(\)}_n = (\)_n - 1/2\Delta(\)_n - 1/12\Delta^2(\)_n \quad \text{Unlagged Variable} \quad (\text{VII-35})$$

$$\tilde{(\)}_n = (\)_n + 1/2\Delta(\)_n + 5/12\Delta^2(\)_n \quad \text{Lagged Variable} \quad (\text{VII-36})$$

On the basis of a limited number of simulations the approximation of second order algorithm by

$$\tilde{(\)}_n = (\)_n - 1/2\Delta(\)_n \quad \text{Unlagged Variable} \quad (\text{VII-37})$$

$$\tilde{(\)}_n = (\)_n + 1/2\Delta(\)_n + 1/2\Delta^2(\)_n \quad \text{Lagged Variable} \quad (\text{VII-38})$$

is proposed.

The integration algorithm programming of the two forms of computation considered is therefore identical*. The programming of assignment of independent

*The term $\Delta v_n \Delta^2 \theta_n$, of second order, is replaced by $\Delta v_n \Delta^2 \theta_n$.

variables for multi-transfer may be considered identical in the two calculation types if $[-(\Delta\theta + \Delta^2\theta)]$ is included in the set of programmable independent variables for multi-transfer. Note that, either $\Delta y \neq [-(\Delta\theta_n + \Delta^2\theta_n)]$, in which case $[-(\Delta\theta_n + \Delta^2\theta_n)]$ should be single transferred, or

$$\Delta y = [-(\Delta\theta_n + \Delta^2\theta_n)] \quad (\text{VII-39})$$

in which case $\Delta v_n + \Delta^2\theta_n$ should be single transferred. It may be deduced that a generalized unit capable of both calculation types is programmable within the framework of that required for the sum of integral calculation type, adding somewhat to diode requirements but not flip flop requirements. In the two output QDD³A, the valuable added facility for three multi-transfers to R_1 and one multi-transfer to R_2 (a modified allocation of multi-transfer units) does not affect the mechanization structure for transfers involving feedback output.

7.3 QDD³A DESIGN FOR INPUT PROCESSING - INTERNAL COMPUTATION TASKS OF FULL AEROSPACE MISSION - The computation task for a full aerospace mission involves capability of executing input processing as well as internal computation. A QDD³A designed solely for several bit increment computation has a degree of input processing capability determined primarily by chosen multi-increment bit length and parallel computation features. Analysis of Chapter X indicates that internal computation requires no more than several bit increment computation at even the modest iteration rates imposed by large program and considerable time shared operation for input processing. In a practical sense then, it is wasteful to mechanize more than three-bit computation for internal computation. We seek to apply the concepts of Chapter IV to the developments following it. The mechanization requirements for input processing are very different from those for internal computation at all but very high iteration rates for input processing assuming intermediate

rate for internal computation. A time shared design for input processing at high iteration rate, chosen for mechanization economy, meets accuracy requirements (with several bit increment computation) provided the input processing program is very small so that the high iteration rate can be achieved without imposing intolerably low iteration rate for internal computation. In view of the considerable numbers of routines of character intermediate between input processing and internal computation (such as air data computations), which are not practical at high iteration rate, the ability to handle a number of demanding computations at intermediate iteration rate has real value. Two new approaches have been investigated in achieving the total set of appropriate computation sophistications in a single computer so that iteration rate (and accuracy) is maximum for a given level of mechanization complexity. The first approach is introduced in Chapter IV and further discussed in Chapter VIII, namely, the double/single precision computation capability in which two 3-bit increment multipliers capable of parallel computations at twice the rate of serial computation may, by programming, also act at designated word times as a six-bit multiplier for "double" precision (as required at intermediate iteration rates, for example, in air data calculations). With this design feature, demanding calculations (usually comprising only a small part of the total program but too numerous to be included in high rate input processing) may be executed with required precision without slowing down the rate of internal computation in the same computation loop, which has program extent for full aerospace mission in certain cases of hundreds of integrators. The second new arithmetic unit design approach derived in Chapter XIII, should not be regarded as essential to efficient QDD²A design. The approach utilizes however, the basic organization features of the QDD²A with second difference computation. The design presented in Chapter XIII requires that inputs be single increment second differences. In applications where this is consistent with scaling the derived multiplier, the D² multiplier, (having

approximately the complexity of a 3-bit increment multiplier) can act as a many-bit increment multiplier. For example, an analytic function such as $\sin \omega t$ can be computed in 10-bit increment steps (for $\omega = \frac{2^{-10}}{t}$) and one part in one million accuracy using the D^2 multiplier. For aerospace applications, however, certain problems are anticipated in exploiting the principle on which the D^2 multiplier design is based. Here the major application of system value would be input processing, however, the digital analog form of inputs for pulse stream converters and the noise and rate character make the predication of single increment second differences of sufficiently fine resolution uncertain. The nature of high rate incremental pulse stream input information representing first differences itself leads to two-bit second differences by any direct conversion method for constant rates. The approach of forcing the introduction to the digital computer of single-bit second differences (by a specially designed servo type preprocessing unit) presents the introduction of non-linear lag effects which can introduce significant errors degrading effective computation algorithm.

Two fundamental approaches to exploiting the principle of the D^2 multiplier in real time computers with sensed inputs are presented. One calls for the design of a hybrid multiplier which is partly conventional for lower significant increment bits and similar to D^2 operation for the remaining increment bits. Provided input signal and noise are scaled correctly, then a two-bit or three-bit second difference input could be computed without preprocessing. The D^2 multiplier does not generalize directly for two-bit second differences in a mechanization of practical value. The second approach to exploiting the principle of the D^2 multiplier is the generation of second differences by second-differencing of whole word sampled values. A sampler unit of sufficient word length will generate single-increment second differences of given resolution, provided the input to the sampler is consistent with their scale. For a significant increase in resolution over that of a

conventional 3-bit multiplier of the same cost as the D^2 multiplier, the sampler word length must be typically of $> 16 \text{ bits} \approx > .04 \times 10^{-4}$ since, for example, at 100 iter sec. (intermediate rate) the QDD²A with conventional multiplier can compute with 10^{-4} resolution on typical high rate inputs where second differencing of inputs requires 2 additional bits to eliminate round off generation of non-single bit second differences inputs. With the double precision feature, which couples two conventional multipliers, a resolution of 10^{-5} is assured at intermediate iteration rate and 10^{-6} high rate loop. Existing sensors and transducers for real time computation with voltage or current sampling typically have resolution of $< 10^{-4}$ and shaft encoder conversion with resolutions of 10^{-4} to 10^{-6} . Therefore the D^2 multiplier offers possible hardware saving in arithmetic unit complexity on the basis of input characteristics, primarily, when coupled with encoder converters. More specifically, a significant gain would probably be effected only with ultra high precision angle measuring devices such as the Microsyn for star tracking systems. As a result of the trend to single telescope systems, the measurement of stellar intercepts are discontinuous from one star to the other. These intercepts represent first differences in navigation error correction computations, therefore, the second differences are discontinuous. This presents another problem in utilizing the D^2 multiplier (but not the conventional multiplier) in the apparently most attractive area of application. The cost of forming second differences is comparable to the difference in cost between a 3-bit and 5-bit conventional multiplier. The considerations of reliability of operation in systems subject to high frequency electrical transients serve to limit the D^2 multiplier application. All of the factors imply effective loss of resolution of the sensor in D^2 multiplier computation by a factor of 2 to 3.

Where input processing is a fundamental computation task in the application, the firmest basis of system design in the light of this is use of the single/double precision arithmetic unit or a conventional multiplier / D^2 multiplier combination in double precision for input processing calculations.

CHAPTER VIII

TYPES OF PROGRAMMABLE MODAL ACTION OF THE FULL SCALE INCREMENTAL COMPUTER IMPLIED BY COMPUTATION TASK AND MECHANIZATION FACTORS

8.0 PURPOSE OF PROGRAMMABLE MODAL ACTION - The major factor in the selection of a particular programmable modal action is the quantitative computation capability attained for a given level of mechanization complexity implied in doing so. The underlying prerequisites for system task, including input processing and internal computation (analyzed throughout major portions of this report) are assumed here, and analyzed in relation to the underlying hardware characteristics. This makes the total set of these functions possible by modal action of arithmetic module and internal communication processes of the basic differential processing unit, the QDPU.

8.1 ARITHMETIC MODULE MODES

A. General Factors Implying Arithmetic Module Modes - Consider the general relations of a computer with arithmetic modal features to conventional computers. If we consider lack of continuous full use of arithmetic capabilities of given hardware in a computer a design deficiency, then a clear deficiency of contemporary incremental and general purpose computer designs exists. The deficiency stems from their processing rate and precision inflexibility for subroutines presenting different types of computation demands. For example, while the G. P. is designed to carry out a computation at fixed rate with an accuracy of 10^{-8} , a computation requiring only 10^{-4} accuracy can be carried out at no higher rate. A computer with the same arithmetic complexity could be designed to achieve twice the processing rate for such routines. But since the other portions of computation task require the higher accuracy,

the conventional rationale is to accept the loss of efficiency.

Actually, since the typical application requires few high accuracy or difficult computations and a majority of low accuracy or easy computations, the loss through functional inflexibility is generally large. The historical lack of problem attack on this design defect has two partial explanations with regard to design philosophy:

1. The widespread impression of the fundamentality of the arithmetic unit; i. e., that tampering with it in design efforts except as a unit cannot be done.
2. The possibility of modal or switching action costing more to implement than the savings gained.

Actually, in answer to (1), the fact that double precision can be programmed in a G. P. (though with inordinate processing rate loss) for 10^{-16} , for example, instead of 10^{-8} accuracy, shows that a 2M bit accuracy multiplier is no more fundamental than two individual M bit accuracy multipliers. With regard to (2), it is shown in later analysis that the double/single precision programmability in the QDD A costs little more than a flip-flop while doubling effective processing rate in lower accuracy computations. Using conventional parallel design principles to equal this rate, the hardware cost would be nearly an order of magnitude higher than by the arithmetic modal design.

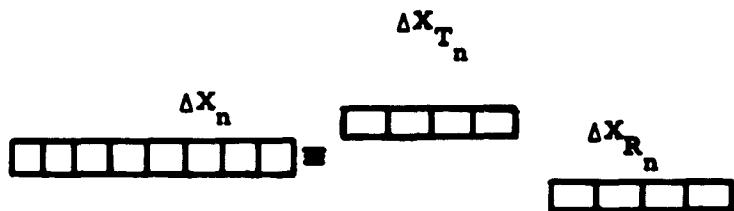
In the QDD A⁸ the advantages of modal arithmetic action are exploited in a natural and efficient manner because the external communication structure between QDPU and external inputs is complete and adequate for double precision as well as single precision computation. The modal operation of pairing several bit transfer units

to act as a many bit transfer device has been evaluated for both conventional multiplier unit designs and the new Δ^2 multiplier developed in the chapter on logical design. In reference to the drum memory case, the transfer for the added significant bits of the independent variable selectively stored in the double precision mode in a flip-flop already present is properly scaled by several bit time delays. These delays are obtained by selective feed in of bits to flip-flops for the second single precision transfer. A coupling of one conventional B bit transfer and D^2 multiplier (of the same complexity which often acts as a 4 or 5 bit multiplier) is described in the following paragraphs.

- B. Transfer Operation in Double Precision Mode by Coupling of Conventional Multiplier and the Developed D^2 Multiplier - The conventional multi-transfer mechanization for ΔX which is 4 bits (including sign) and the developed D^2 multiplier have essentially the same flip-flop requirements. When the second difference $\Delta^2 X$ is known to have a sufficiently small maximum the D^2 multiplier can act as a many bit multiplier; e.g., a sinusoid of frequency f , constant angular rate, can be computed with D^2 multiplier at iteration rate (IF), with $\Delta^2 X$ of $\log_2 (IR/2\pi f)$ bits not including sign, hence at 100 iter/sec, a .125 cps sinusoid can be computed with an 8 bit ΔX (including sign) as compared to the conventional 4 bit ΔX multiplier of about the same complexity. Input processing presents insurmountable granularity problems to a conventional DDA, and except at very high iteration rates also to a several bit increment DDA. Thus, in input processing, the value of many bit increment computation is evident. A programmable double precision mode has the value of meeting this need for

demanding computation routines. This makes possible double rate computation for the majority of computations which are not so demanding but largely make up the bulky program for full aerospace mission. Input processing, especially at low rate, is characterized by comparatively large $\Delta^3 X$ max/ ΔX max ratio. The large ratio implies that the D^2 multiplier may be dependent upon attaining in the worst case only several bit increment computation. Double precision mechanization studies show that the two D^2 multipliers do not readily couple to increase resolution by more than one bit. On the other hand, a conventional 3 bit increment and D^2 multiplier can be coupled to allow for 4 bit increment in $\Delta^3 X$, increasing the multi-increment precision of a double precision mode by a factor of 16. Consider the representation of a many bit ΔX in terms of truncated and residue components:

$$\Delta X_n = \Delta X_{T_n} + \Delta X_{R_n}$$



4 bit residue case

For a $\Delta^2 X$ which is known expressible as a 4 bit residue, it is seen:

$$\Delta^2 X_n = \Delta^2 X_{R_n} \quad (\text{VIII-1})$$

Updating of ΔX_n is:

$$\begin{aligned} \Delta X_{n+1} &= \Delta X_n + \Delta^2 X_n \\ &= \left[\Delta X_{T_n} + \Delta X_{R_n} \right] + \Delta^2 X_{R_n} \quad (\text{VIII-2}) \\ &= \Delta X_{T_n} + \left[\Delta X_{R_n} + \Delta^2 X_{R_n} \right] \end{aligned}$$

A single carry C_n results in adding residues, hence:

$$\Delta X_{R_u} + \Delta^2 X_{R_n} = \Delta X_{R_{n+1}} + C_{n+1} \quad (\text{VIII-3})$$

Thus, the equality:

$$\Delta X_{n+1} = \left[\cdot \Delta X_{T_n} + C_{n+1} \right] + \Delta X_{R_{n+1}} \quad (\text{VIII-4})$$

Where the integrand is y and the algorithm corrected quantity involved in transfer is \tilde{y} , then:

$$\tilde{y} \Delta X_{n+1} = \tilde{y} \cdot \left[\Delta X_{T_n} + C_{n+1} \right] + \tilde{y} \Delta X_{R_{n+1}} \quad (\text{VIII-5})$$

is to be transferred for double precision. A conventional 3 bit multiplier can transfer $\tilde{y} \Delta X_{R_{n+1}}$. The D^2 multiplier can transfer

$\tilde{y} \Delta X_{R_{n+1}}$ The D^2 multiplier can transfer $\tilde{y} [\Delta X_{T_n} + C_{n+1}]$ employing C_{n+1} in the same manner as $\Delta^3 y$ in single precision mode. The conventional and D^2 operate as a coupled double precision multiplier by programming integrands to be identical and delaying start of the D^2 multiplier by 4 bit times. The latter is effected using the 8 update mechanism in which certain 8 quantities, being updated before others (with savings in channel and arithmetic requirements), are stored in the communication core memory (total core count 10 to 12 words). The time of drawing out the core stored 8 quantities differs in double precision from that of single precision by 4 bit times.

8.2 INTERNAL COMMUNICATION MODES OF THE BASIC PROCESSING UNIT (QDPU) - The evaluation of the QDPU internal communication modal design approach is more complicated than the obviously powerful arithmetic modal design approach analyzed in the preceding paragraphs. Evaluation should be relative to conventional parallel DDA designs since only a parallel processing unit can match the QDPU in computation capability, the level of which is established as necessary for pertinent applications. Since the QDPU carries out several operations together using internal communication (within the QDPU rather than QDDA as a whole) while a parallel DDA integrator does not (within the DDA integrator but rather within the DDA as a whole), the degree of overall external communication is expected to be less. In application programming analyses, the number of external outputs (counted once) communicated is at most one-half that of a parallel DDA with comparable program capacity (but much lower computation capacity than the QDDA); also, though less useful it is found that input sinks selecting outputs is at most three-quarters that of the DDA. Rapid access (core) memory for communication is

reduced by one-half on this basis, and another factor of one-half or more using the new second difference communication developed in the chapter on multi-increment computation, with communication selection requirements for full communication reduced by more than one-third. Thus, substantial hardware savings are made possible by internal communication processes of the QDPU and serve to compensate the modest cost of these modal features. A second important hardware saving made possible by internal communication modality is the saving in the number of registers. For pertinent comparison, the parallel DDA without division algorithm, requires; with drum memory, three extra channels (for register information) and associated read write update logic, or with core memory, over one hundred and fifty words more of core memory.

Because of the large gains in computation capability made possible by reduced R-register count and quotient algorithm in the QDDA, a conventional parallel DDA with actually matched computation capability would require a higher iteration rate obtained by further parallelity and further cost than indicated above.

8.3 DECISION OPERATIONS IN THE MECHANIZATION OF MULTI-INCREMENT QDDA WITH SECOND DIFFERENCE COMMUNICATION

A. Introduction - It is essential that the QDDA have the decision capabilities of the conventional DDA in order that supervision by G. P. be nil or held to a minimum in decision modified calculations. As a result of the special features of second difference communication the decision mode design problem is quite different than in a conventional DDA, though, of course, the fundamental approaches in generating decision modified functions must be used. In the conventional DDA, conditional cutoff of an integration with respect to time is readily produced by replacing time by decision function (programmed as integrator input for independent variable). In

the QDDA the decision function cannot be regarded as a normal input since the latter is generally a second difference, not directly used in multi-transfers. Secondly, if the multi-transfers to R were conditionally cut off as possible in a conventional integrator, the output cutoff does not cut off integration in the QDDA since the QDDA outputs are second differences. Despite the problems indicated by these initial observations a decision capability was designed into the QDDA which on balance actually increased the "integrator" equivalent of the QDPU relative to conventional DDA over the estimates previously based on continuous operations. The modest hardware requirement to accomplish the decision features is comparable to that in the conventional DDA.

- B. Decision Command Generation Parallel with Input Processing -
- The generation of a decision command function of the simplest type requires only one DDA integrator. In order to retain QDPU equivalent to four plus DDA integrators, the decision command functions are best generated in QDPU, which are assured of performing a substantial task in generating some other computation for output of the parallel channel. Since decision command signals are generated by outputting the sign of a y register clearly no multi-transfer action is used. This suggested the QDPU in an input processing mode which, in generating the sum of two multi-increments of double precision, utilize the entire instantaneous arithmetic capability of the computer. Clearly, if the decision command function of the simplest type is generated in parallel with input processing calculations the integrator equivalent has increased by one "integrator." Since input processing generally involves at most two internal computer generated inputs (and two external inputs) the programming of a test function as a third input is impossible in

normal programming structure. The test function x becomes d^2x followed by normal δ programming structure. The test function x becomes d^2x followed by normal δ register and y register updating where in the simplest decision type y was initialized at $-x_0$, which then leads to decision function output D of $D = 1$ when $x > X_0$, $D = 0$ when $X < X_0$. Since input processing is executed at high rate, and internal computation at moderate rate, test variable d^2x actually enters the high rate loop at the low rate. Consider for example, the case where the rate ratio satisfactory for input processing is 4 (though in double precision the use of intermediate rate is more typical). In this case, the effective scaling of d^2x to x is increased by 4 since dx updates x at 4 times the rate of the internal computation loop. The decision command variable generated at high rate is used only one fast iteration out of four in the internal computation loop. The multi-iteration rate feature causes no difficulty in generating decision command signals for internal computation or input processing. The operation is essentially free in the simplest decision command type because the processing rate is unchanged when full arithmetic capability is devoted to input processing.

Input processing and decision command operations are executed in parallel when a decision command operation is required in the program code of the QDPU. Since the decision generation does not need feedback to the δ_b register as required by the algorithm for multi-increment computation with second difference communication, which is used in input processing (and internal computation), the δ_b register is available for use in input processing. The input program code has two bits which determine the δ register in which is placed an input of given address in rapid access communication memory. Since, normally, only three δ registers

have programmed inputs the normal code can select an input to δ_b with elaboration of the code. In the case of input processing, this fourth input register has definite utility because input processing can call for, on occasion, the combination of disparate variable pairs in(usually)pure integration processes. In the case of internal processing, the extensive programming studies show that only three input δ registers are required for essentially full versatility. Since the δ_m register is required for algorithm purposes, a saving of one channel results together with the input arithmetic requirements. The latter comment presumes that no additional arithmetic requirement is necessary. The δ_m register, during input processing, is applicable only if a single input is allowed. In navigation equations and thrust cutoff, this limitation presents no difficulty. Actually, if programing limitations were encountered the unused output mechanism of the decision channel to accomplish multi-input capability to δ_m without cost in flip-flops would be relatively simple.

- C. Decision Response Modes - Upon absorption of decision command signals in the QDPU, decision action of conditional transfer takes place using simple logical "and" operations. Absorption of decision command signals into the QDPU has a simple mechanization which obtains highly efficient QDPU operation. The total number of analytic variables (as distinguished from decision variables) which are concomitantly absorbable is reduced only one, from eight to seven, while what is most important, the analytic variables may be collected in all three registers. When the QDPU has decision programming bits (two in number) indicating it is to have action in one of several alternative decision modes, the address of

inputs to the δ_m register determines whether one of two or more variables is to be treated as a decision variable, the remainder to be treated in the normal manner for analytic variables in updating the δ_m register. The decision command bit D is then used as the conditioning variable for any of the several alternative conditional transfers of δ_m contents to y_m , y_1 registers. All conditional operations obtainable in conventional DDA may be obtained by conditional transfers:

1. δ_m to y_m unless $D = 0$, which enables function limiting.
2. δ_m to y_1 unless $D = 0$, and unconditional transfer to δ_m , which enables cutoff of updating of a variable with respect to one of its components.
3. δ_m to y_m if $D = 1$, $-\delta_m$ to y_m if $D = 0$.
4. $\pm y_1$, δ_m transfer to R according as $D = 1$ or 0, enabling sign control.

The fact that any one of the simply mechanized decision response modes conditions the otherwise ordinary fully programmable action of the QDPU implies an integrator equivalent of the unit that is greater than the average during such modes. Examples of programming the QDPU in decision modes for doppler damping and thrust-cut applications are analyzed in the chapter on application programming and evaluation of the QDDA.

8.4 MULTI-ITERATION RATE FOR INPUT PROCESSING AND INTERNAL COMPUTATIONS - The development during this contract study of an arithmetic module, capable of alternating the work load between input processing (double precision) and internal computation (single precision), enables a marked savings in hardware by time sharing. The basic computation

requirements of input processing and internal computation may imply not only different multi-transfer requirements, but in certain critical input processing applications may imply different iteration rates. The QDDA can be programmed to obtain single and double precision as desired. To obtain efficient different iteration rates for the two types of calculation for a drum memory computer, the appropriate allocation of read and write heads and some switching logic is required. Input processing calculations which require a relatively high iteration rate derive this requirement not only from the high frequency character of the computer inputs, but also from a high computation error sensitivity where high accuracy is required. In general, computation error sensitivity stems from computation routine metastability or long term instability; i. e., the accumulative rather than damped error response. Because metastability and instability stem from a feedback character of the calculations, it is generally true that a sensitive input processing calculation is essentially isolated in generation from internal computations, which in essence simply utilize the results of the former. Evidently, a sensitive input processing, requiring a relatively high iteration rate M times that of the internal computation need for acceptable accuracy, probably only communicates with the latter at the lower iteration rate with instantaneous rather than accumulated outputs. Thus, in cases where the QDDA has a multi-iteration rate input processing-internal processing operation, the communication setup between one and the other processings probably could be chosen, with a modest hardware saving, to use the same rapid access memory setup as at equal rates (though with program scale changes of inputs from input processing QDPU in internal computation). However, performance is assured by supplementing the communication rapid access memory by a half dozen 4 bit core registers for accumulating outputs of input processing in a high rate loop. A QDDA with drum memory can be set up for the multi-iteration rates by alternating print instructions to write heads, spaced M

ratio of delays from read heads, while alternating QDDA modal action at half the lesser delay interval of that of input processing and internal computation lines.

8.5 MODAL TYPES SUMMARY - The following modal types discussed in detail in previous paragraphs are developed for the full scale computer:

- A. Iteration rate mode is either high or intermediate accordingly as the QDPU number is ≤ 14 or > 14 . Input processing is presumed to involve ≤ 14 QDPU (consistent with application study).
- B. Arithmetic mode is single or double precision for any QDPU according to the arithmetic mode programming bit. Generally, input processing uses double precision at high iteration rate. Certain error sensitive calculations such as sinusoid calculation on external input angles may be computed with required precision at intermediate rate using double precision.
- C. Internal Communication Modes of the QDPU
 1. Transfer allocation modes are those selected in QDDA computation programming studies and delineated in the programming code analysis.
 2. Decision command mode is automatic in one channel of QDPU used also for input processing. Accordingly, as a programmed variable exceeds a given constant (as determined by the sign of a given δ -register), the output is the decision command signal 1 or 0 used in decision response modes. Up to 14 independent decision command signals can be generated without cost in overall processing rate.
 3. Decision response modes, according to the two decision response programming bits, interpret two or more inputs

to a given δ register (δ_m register) as a decision signal or normal variables input depending upon whether or not the source was the second channel of the first 14 QDPU. The updating is in normal fashion with normal variables and the interpretation of the decision command signal (a normally programmed input) is according to programming bits, and the decision response programming bits, of the QDPU for which the actions are one of the following:

- (a) Transfer Conditional Inhibition of δ_m to y_m , thus enabling function limiting.
- (b) Update Conditional Cutoff of δ_m to y_1 and unconditional update of y_m by δ_m .
- (c) Transfer Conditional Sign Change of δ_m to y_m .

CHAPTER IX
DDA AND QDD³A SIMULATIONS ON THE IBM 704
COMPUTER AND PRIMARY RESULTS

9.0 OBJECTIVES AND PRIMARY RESULTS ON THE DDA AND QDD³A SIMULATION EFFORTS - Approximately one third (11 hours) of the total allocated programming hours on the IBM 704 for the Phase II effort was utilized in simulations of ordinary and elaborated conventional ternary DDA and of the QDD³A. The broad objective of this effort was to assist in the development and evaluation of incremental computer designs for internal computation. Internal computations account for the majority of the integrators utilized in typical aerospace application programs, but generally do not involve, in fullest measure, the special computation problems of input processing explored during Phase I. The full aerospace mission has associated programs requiring a DDA with a capacity of several hundred precision integrators, which implies that, while the individual internal computation routine might not in all cases be challenging nevertheless the overall computation task does challenge the most sophisticated existing DDA computers.* The reduction of mechanization complexity in the computer system designed to handle input processing and internal computation was ultimately obtained through a single time shared basic processing unit (generalized integrator), which basically implies a need for further increased internal computation capacity (as the result of the time sharing feature). The choice of specific simulation efforts in internal computer design is made most efficient by concentrating effort on those types of internal computations that occur in aerospace applications and that present the basic source of limitations of

*The goal of reducing GP computer mechanization requirements in a GP-DDA system to a fraction of that required in previous aerospace systems requires for internal computation a DDA of >250 integrator capacity.

computation capacity of existing DDA computers. Aerospace applications were shown to imply the capability of executing computations involving division. The lack of sufficient quotient capability is one of the two basic limitations of the conventional DDA. Division operations are generally recognized to present severe accuracy limitations. The major previous real time application of the DDA, airborne (pure) inertial navigation, has generally avoided division altogether by making programming use of the narrow altitude and velocity range of the conventional aircraft. The lack of precision division capability is becoming apparent in the latest airborne doppler damped inertial navigation systems where long term navigation accuracy is sought. Precision division capability is a major requirement in full aerospace applications. The second basic limitation of the conventional DDA in internal computation is common to the fundamental limitation of input processing, as it is directly related to rate limit and resolution limitations that generally lower overall computation capacity and actually accentuate the division operation limitations as well as the other typical computations that are applications of integration in the DDA. In recognition of these two basic limitations of existing DDA computers, the programming effort allocated to DDA and QDD³A simulation studies was directed primarily toward investigations of division and multi-increment DDA design techniques and their evaluation. Simulations of sinusoids were also executed to complement efforts of phase one. The most important simulation result of the overall effort was verification of the QDD³A multi-increment quotient algorithm developed during the second phase of the program. A second result may potentially have value in slightly elaborated conventional DDA, but could not be thoroughly simulation evaluated within the scope of this effort. This result was a discovery of a technique for improved digital Stieltjes integration. Realization of this technique could improve quotient capability of a near conventional DDA. As an aerospace full mission computation program executed largely by QDD³A implies a very large step in

increased computation capacity in relation to existing DDA computers, the major analytical development of design techniques for a multi-increment quotient algorithm DDA was chosen as the primary simulation evaluation program objective during the limited period prior to the required programming effort for evaluation of the strap-down processor constructed during Phase 2. This development occurred mid-course during Phase 2. Successful simulation of the computer that has been referred to as the QDD³A has confirmed that a major breakthrough in DDA design technique was accomplished.

9.1 PROGRAM FOR SIMULATION OF DDA COMPUTATIONS AND THE MODIFIED PROGRAM FOR QDD³A COMPUTATIONS

- A. Programming Approach - Simulation of computations involving DDA integrator ensembles with elaborated DDA integrator designs involves programs that may be basically closely related, but which, if not provided for in the programming approach, are not readily altered from one simulation to the next. To minimize the programming effort in preparation for successive simulations, a program for general integrator ensembles of DDA integrators with all the alternative algorithm features was developed. Thus, a single program modification card could supply the format for the next simulation. When the latter QDDA was analytically developed, the problem of programming for simulation evaluation was obviated by altering the DDA program to treat a set of DDA integrators storagewise as a single QDDA and incorporating the several basic changes of operation.**
- B. Program Structure for Simulation of an Arbitrary Computation by a System of DDA Integrators (with mixed Higher Order Algorithms, Derivary Communication, Roundoff Reduction and**

Update
each
Iteration

Fixed
Throughout
a Simulation

Fixed
Subroutine
for all
Simulations

Special Initialization Features) - A program structure was developed for general DDA system simulations. The processing of a single DDA integrator and execution of its communications through a single iteration of an arbitrary computation can be completely defined in the following general manner:

1. State a set of numbers which define:
 - a. State of the integrator before processing
 - b. The incoming communications to the integrators for that iteration.
 - c. The special features of operation of that particular integrator, namely, integration algorithm, roundoff reduction features.
 - d. Identity of the other integrators to which the said integrator communicates, and the type of utilization by the integrator communicated to (i. e., integrand and/or independent variable) and the sense of utilization of these communications (i. e., direct or with sign reversal).
2. Subject the set of numbers described in 1 to a subroutine (called the integrator processing subroutine) which computes (using 1a, 1b, and 1c). a) the new integrator state, then updates 1a; and b) the new integrator outputs, then (using 1(d), updates 1(b) of the appropriate other integrators. The subroutine will at its end replace the input communication runs for "immediate use" (by then, already used)

with the input communication sums for "subsequent use" (to be used at the next iteration, when further communications have been summed into it).

A program capable of carrying out an arbitrary computation involving integrators is obtained as follows:

3. Storage blocks, N in number, are allocated in the rapid access memory. Each storage block contains information of ia, ib, ic, and id associated with a single integrator. The initial values of the numbers in each storage block are inserted in the rapid access memory prior to simulation start by means of cards or tape.

After program start, the rapid access storage blocks are updated during execution of the program (i.e., ia and ib) numbers are updated in the manner indicated below.

4. The operations of 2 are incorporated in the integrator processing subroutine.

5. Lists of the specific data types and operation of I and II are:

a. Integrator data storage block information:

(1) y-register content (whole word)

R-register content (whole word)

(2) Input communications summed in a space for immediate iteration use and next subsequent iteration use, respectively, for each quantity:

Independent variable communication total:

1st order for immediate use: $\Sigma \Delta X$

1st order for subsequent use $\Sigma \Delta X$

2nd order for immediate use: ΣD_x

2nd order for subsequent use: ΣD_x

Integrand variable communication total:

1st order for immediate use: $\Sigma \Delta_y$

1st order for subsequent use: $\Sigma \Delta_y$

2nd order for immediate use: ΣD_y

2nd order for subsequent use: ΣD_y

At the end of integrator process the subsequent sums replace the "immediate use" sums.

(3) S_1 Integration Algorithm scale factor for 1st order term (whole word)

S_2 Integration Algorithm scale factor for 2nd order term (whole word)

T_{OI} Test number used in decision for overflow inhibition and R-register reset to 1/2 (whole word)

(4) F_r^{1st}, F_r^{2nd} ($r = 1, 2, 3, 4$): Factor by which associated communication is multiplied before being sent to other integrator data block.

C_r^{1st}, C_r^{2nd} ($r = 1, 2, 3, 4$): Storage places (in other integrator data blocks) for presently processed integrator outputs (1st, 2nd order terms) to be sent appropriately for correct: (1) Integrator (2) Type of utilization to be made (integrand or independent variable) (3) Time of utilization (as soon as the integrator is processed or in the subsequent iteration).

- b. The integrator processing subroutine carries out the following operation:

$$(1) \text{ Y-register updating: } Y_n = y_{n-1} + \sum \Delta y_n \quad (\text{IX-1})$$

- (2) Integration algorithm updating of R-register with overflow inhibitor test feature:

Reset R-register if $|y_n| < T_{OI}$ to $R_n = 1/2$

Otherwise,

$$\Delta R_n^* = \left(y_n + S_1 \sum_{c=1}^C \Delta y_n + S_2 \sum_{c=1}^C D_n \right) \Sigma \Delta X_n \quad (\text{IX-2})$$

is added to R_{n-1} to obtain R_n^*

If $R_n^* > 1$, then take $R_n = R_n^* - 1$

$R_n^* < -1$, then take $R_n = R_n^* + 1$

$-1 \leq R_n^* \leq 1$, then take $R_n = R_n^*$

- (3) Overflow to be communicated as 1st order terms:

If $|y_n| < T_{OI}$, then No overflow: $O_n = 0$

Otherwise if:

$R_n^* > 1$, then take $O_n = 1$

$R_n^* < -1$, then take $O_n = -1$

- (4) Communicated 2nd order terms

$$D_n^* = \sum \Delta x_n \circ \sum \Delta y_n \quad (\text{IX-3})$$

(5) Scaling using F-factors:

$$F_r^{\text{1st}} O_n = \Delta x, \Delta y; F_r^{\text{2nd}}, D_n^* = D_x, D_y \quad (\text{IX-4})$$

(6) Replace contents of input common for "immediate use"
(already used) with contents of input common for "subsequent use."

C. Simulations of Reciprocal Calculation by Conventional and
Digital Stieltjes DDA and the QDD'A

1. Program Features - It is generally recognized that a conventional DDA executes reciprocal calculations (and more complex operations involving division) with poor accuracy in comparison to most other computations. Alternative programs employ integrators alone (no servo in Amble's method) in the one case and the program for implicit calculation with integrators and a servo in the other case. The alternative methods are recognized to result in different detailed error properties but generally the same error magnitudes. The goal of this study effort was the development of a precision computer. The conclusion that servos involve inherent errors from lag effects and servo mechanism properties led to placing primary study emphasis on integrator systems without the use of servos. It will be seen that the primary error affects in reciprocal calculation, as contrasted to other calculations, result from imperfect digital Stieltjes integration algorithm. The reciprocal calculation by a DDA provides a test calculation for the fundamental process of integration with respect to a variable other than full rate where single transfer effects the operation.

To evaluate conventional DDA with ordinary and elaborated algorithms, short run tests were devised so as to be extraordinarily severe. This was accomplished by generating high frequency inputs with large amplitude excursions requiring that integrator register lengths be very short. The test input function was, in all cases, of the form

$$I_n = A + B \sin \theta_o n \quad (\text{IX-5})$$

for which the DDA was allocated the task of computing $1/I_n$. For $B = 0$ the input function represents generally a partial (less than full) rate. For general A, B the effects of oscillation could be evaluated.

The classical type* algorithm for reciprocal calculation may be derived by either of two methods: (1) Exact numerical difference relations (2) Numerical integration algorithm. While all algebraic computations such as the reciprocal calculation may be assigned an exact numerical difference relation, differential equation solution computations are directly analyzed in terms of numerical integration algorithms. Algebraic equations when differentiated produce differential equations which have algorithms for solution directly analyzed in terms of numerical integration algorithms. The integration algorithm is therefore general in applicability to algebraic and differential equation solution whereas the direct difference relation approach is not. Any computation may be executed using appropriately, for each integrator of the system, a particular one of only two integration algorithms (both occurring in the program). A computer which

*Based on numerical rather than digital processes.

performs exact difference equation solutions and also the integration algorithms would formally require mechanization for four programmable algorithms for second order accuracy. Consider the exact difference relation for reciprocal calculation

$\theta_n = 1/I_n$ as obtained in the two steps of differencing $\theta_n I_n$.

$$\Delta(\theta_n I_n) = \theta_{n-1} \Delta I_n + I_n \Delta \theta_n = 0 \quad (\text{IX-6})$$

and substituting $I_n = 1/\theta_n$ and solving for $\Delta \theta_n$ to obtain

$$\Delta \theta_n = -\theta_n \theta_{n-1} \Delta I_n \quad (\text{IX-7})$$

an exact difference relation. Next consider the differential of θI

$$d(\theta I) = \theta dI + I d\theta \quad (\text{IX-8})$$

in which substitute $I = 1/\theta$ and solve for $d\theta$ to obtain

$$d\theta = -\theta^2 dI \quad (\text{IX-9})$$

An exact difference is formally derived by integrating over an interval from $(n-1)\tau$ to $n\tau$,

$$\Delta \theta_n = \int_{(n-1)\tau}^{n\tau} \theta dx(t) \quad (\text{IX-10})$$

where

$$x(t) = \int_{(n-1)\tau}^t \theta dI$$

Since θ is available only at $t = (n-1)\tau$ and before, the lag correction integration algorithms in virtual variables is appropriate, hence to second order

$$\Delta \theta_n = \left[\theta_{n-1} + \frac{1}{2} \Delta \theta_{n-1} + \frac{5}{12} \Delta^2 \theta_{n-1} \right] \Delta x_n \quad (\text{IX-11})$$

where

$$\Delta x_n = \int_{(n-1)\tau}^{n\tau} \theta dI.$$

The same consideration implies the computation of Δx_u by

$$\Delta x_n = \left[\theta_{n-1} + \frac{1}{2} \Delta \theta_{n-1} + \frac{5}{12} \Delta^2 \theta_{n-1} \right] \Delta I_n \quad (\text{IX-12})$$

That the computation in terms of integration algorithm agrees with the computation with exact difference relation to first order is deduced by substituting Δx_n in the $\Delta \theta_n$ relation:

$$\begin{aligned} \Delta \theta_n &= \left[\theta_{n-1} + \frac{1}{2} \Delta \theta_{n-1} + \frac{5}{12} \Delta^2 \theta_n \right]^2 \Delta I_n \\ &\approx (\theta_{n-1} + \frac{1}{2} \Delta \theta_{n-1})^2 \approx \left[\theta_{n-1}^2 + 2(\theta_{n-1}) \left(\frac{1}{2} \Delta \theta_{n-1} \right) \right] \Delta I_n \\ &\approx \theta_{n-1} (\theta_{n-1} + \Delta \theta_{n-1}) \Delta I_n \\ &\approx \theta_{n-1} \theta_n \Delta I_n = L \theta_n \text{ exact difference} \end{aligned} \quad (\text{IX-13})$$

If account is made for second order differences of virtual and desired variables the agreement is good to second order.

The simulated DDA configuration for reciprocal calculation is indicated in Figure 9-1.

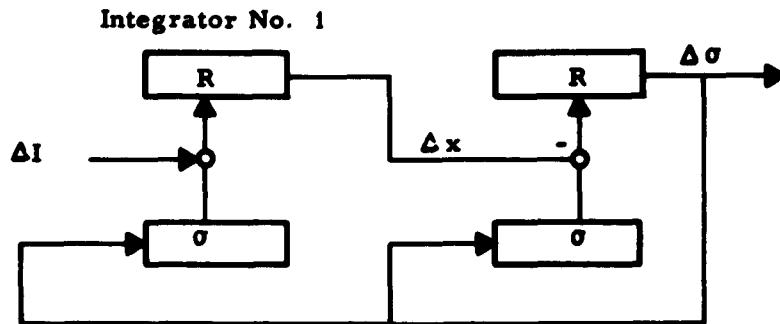


Figure 9-1. Schematic of Simulated DDA Reciprocal Calculation

Programmable integration algorithm (apart from elaborations developed) was of the form

$$\Delta x_n = \left[\theta_n + S_1^{(1)} \Delta \theta_n + S_2^{(1)} \Delta^2 \theta_n \right] \Delta I_n \quad \text{Integrator No. 1} \quad (\text{IX-14})$$

$$\Delta \theta_n = \left[\theta_n + S_1^{(2)} \Delta \theta_n + S_2^{(2)} \Delta^2 \theta_n \right] \Delta x_n \quad \text{Integrator No. 2} \quad (\text{IX-15})$$

2. Initial Single Increment DDA Reciprocal Calculation Runs -

Initial runs simulated a conventional single increment DDA with alternative integration algorithms of the classical form. The reciprocal calculation to be executed was

$$\theta_n = \frac{0.25}{0.50 + \frac{7}{32} \sin(2^{-6} n)} \quad (\text{IX-16})$$

which in a single increment DDA may be carried out with register lengths of five bits, at most, using Amble's method. For a DDA iterating at 200 it/sec, the test calculation involves

frequency components at about 1 cps. Initialization of R and y registers was selected with alternative subsignificant biases to test the effect of small perturbations. Algorithms tested were the same for integrators no. 1 and 2 and included the cases

Run (1) $S_1 = 1/2$, $S_2 = \frac{5}{12}$ Second Order Algorithm

Run (2) $S_1 = 1/2$, $S_2 = 0$ First Order Algorithm

Run (3) $S_1 = 0$, $S_2 = 0$ Lagged (Zero Order) Algorithm

Run (4) $S_1 = 1$, $S_2 = 1$ Led (Zero Order) Algorithm

In these runs the communicated Δy was used as the algorithm Δy directly and the derivative as $\Delta^2 y$. All runs (which were checked for programming accuracy) demonstrated errors in excess of the granularity in 500 iterations and errors of the order of the variation of O_n in 2000 iterations. An approximate graph of results and the desired computed function are presented in Figure 9-2. The general effect consistently observed is an incorrect attenuation of the computed function amplitude. The first three runs were executed first.

The first two runs have lag correction numerical integration algorithms of second and first order accuracy and, in a whole word (rather than single increment) incremental computer, should yield accurate results. The fact that large errors resulted is a consequence of the single transfer-single increment mechanization simulated. Sinusoid computations by single increment DDA yield good results with the second order and first order algorithms (lag adjusted for serial or parallel computation) and poor results for lagged or led zero order algorithm. The DDA reciprocal calculation results for Run 3 were somewhat poorer for the lagged algorithm. Analysis of whole word computation of reciprocal

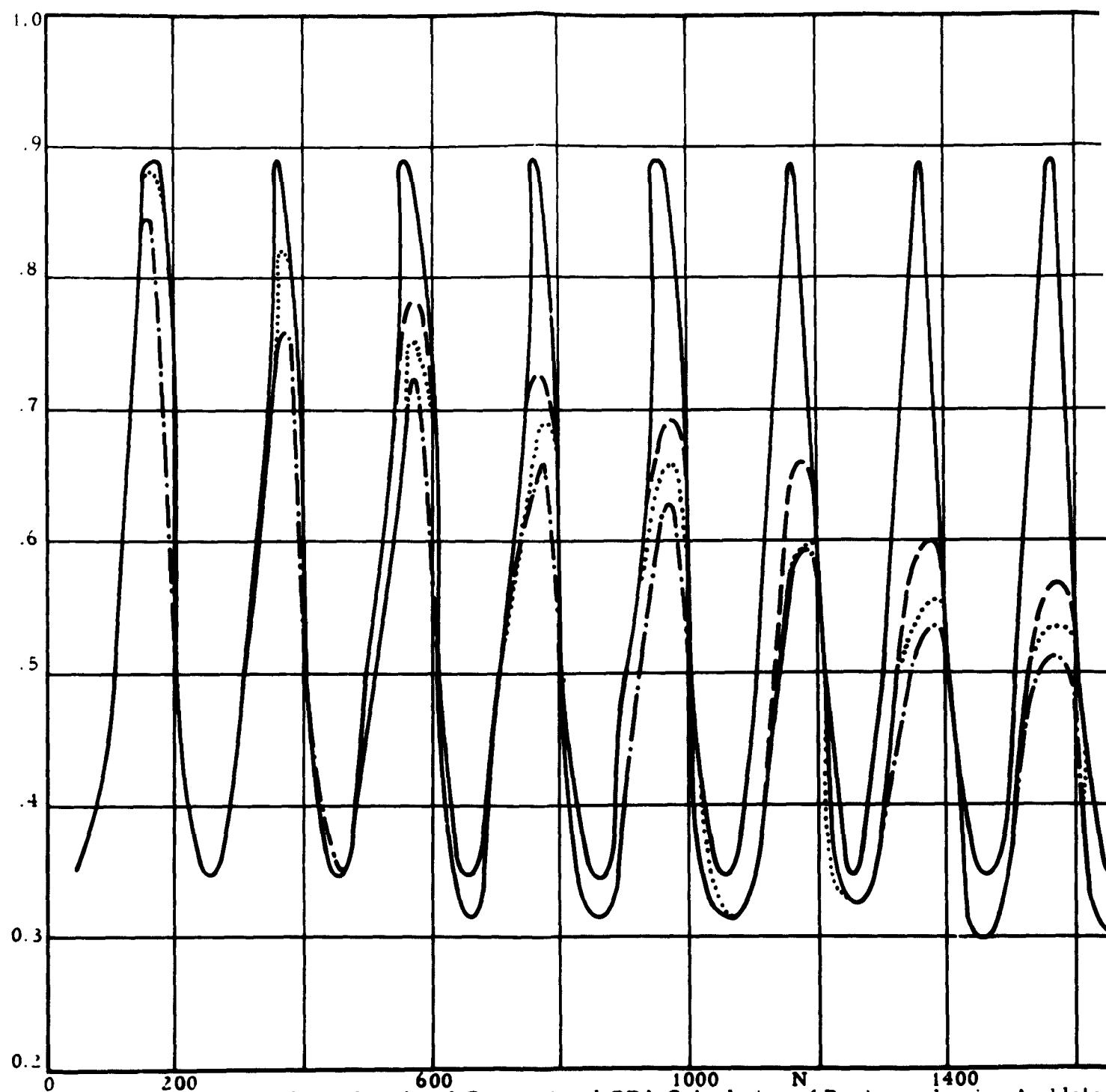
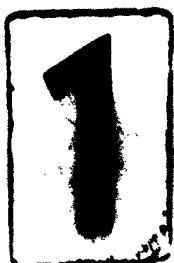
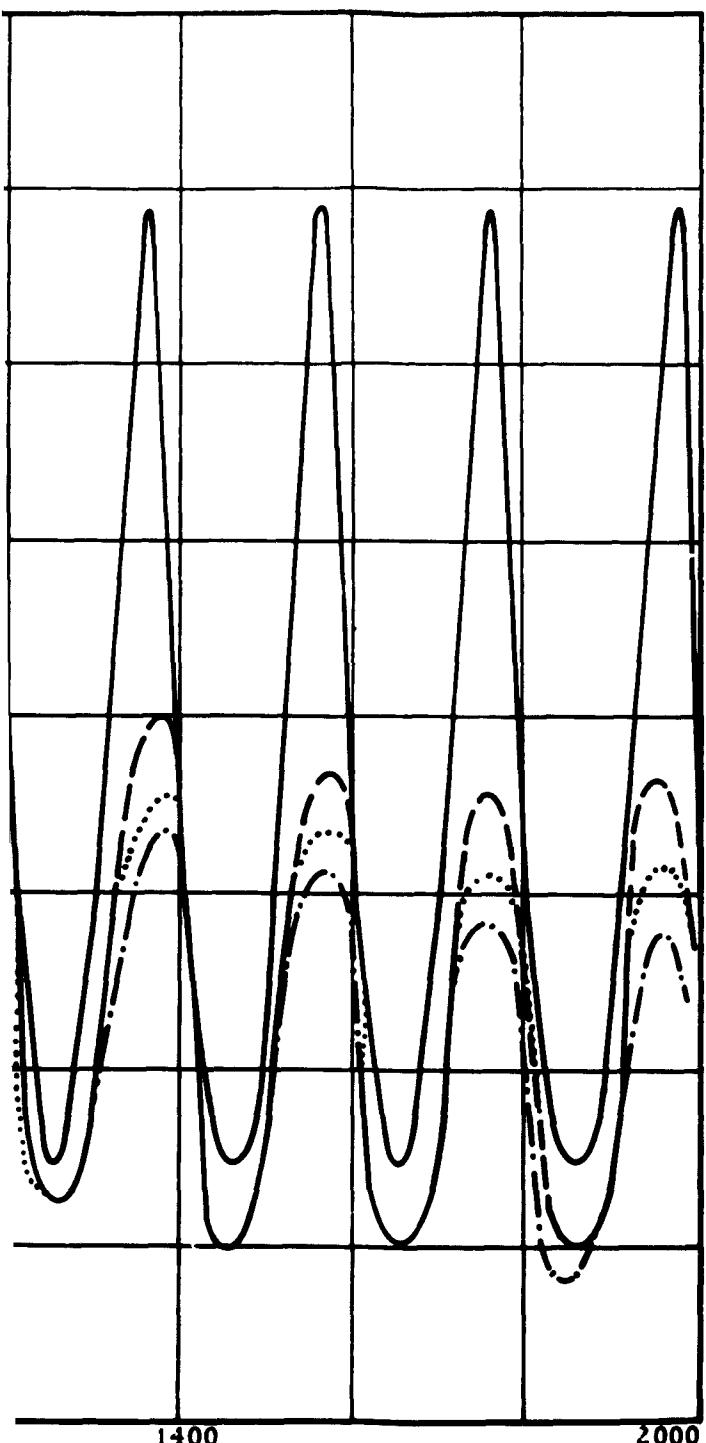


Figure 9-2. Simulated Conventional DDA Calculation of Reciprocal using Amble's 1 Integration Algorithm Classically Mechanized





True
 $S_1 = .5, S_2 = 5/12$
 - - - - - $S_1 = 0, S_2 = 0$
 - - - - - $S_1 = 1, S_2 = 1$

Classically Mechanized Algorithms

$$\sigma = \frac{1}{I} = \frac{1}{2 + 7/8 \sin 2^{-5} n}$$

$$Y_0 = .50292997$$

$$R_0 = .5$$

$$R_0^* = .987306875$$



calculations as well as simulation of single increment computation indicates that lag will produce oscillation amplitude attenuation for the reciprocal calculation with inputs having an oscillatory component. The analysis of the whole word computation case indicated that a perturbation of the observed error magnitude could, for the selected type of input to reciprocal calculation, be generated by a one-half iteration lead. Run 4 was therefore selected to determine the degree of improvement in the single increment DDA using an incorrect numerical integration algorithm which should compensate for another error source. Results showed slight improvement but not of the degree called for on the numerical computation basis. Runs were made with the same set of integration algorithms but with different initial R register settings of $R_0 = 0.5$ and 0.75 and slightly perturbed (subsignificant) initially register values. Results were little changed in the different runs. It was concluded that the error effects in digital Stieltjes integration with single increment are not solely compensatable by any limited modification of classical integration algorithm (mechanized in the conventional manner for DDA) from that called for by theory, but rather require a more subtle basis in mechanization that should be explored in greater detail. That the error effects are attributed to the mechanics of Stieltjes integration (independent variable not time) was a deduction, of course, based on the fact that other simulations of single increment DDA which involved integrations with time independent variable led to good results in all cases consistent with the theory of numerical integration. Detailed study of the micro-aspects of the runs were focused on the reason for the

failure of transmission of integration algorithm terms.

3. Reciprocal Computation by DDA Integrator Setups Other Than Amble's Method - The poor performance in reciprocal calculation by conventional DDA with conventional integration algorithms effected by typical mechanization approaches stimulated the selection of a set of alternative integrator setups and algorithms (the latter with Amble's method) for evaluation in a parallel effort. Because promising results were obtained by the concomitantly developed digital Stieltjes algorithm in Amble's setup, the two alternative integrator ensembles then being programmed were not thoroughly evaluated beyond a single run. One of the approaches gave relatively good results compared to Amble's method with the same conventional algorithm mechanics, but results were not comparable to Amble's method with the new algorithm (later developed). It is possible that if the new algorithm were adopted in this alternative computation for reciprocal calculations, results might be better than with Amble's method in very much longer term operation than that carried out in evaluating the new algorithm. The devised alternative reciprocal computation methods and run parameters are indicated in the diagrams on the next page. The second method, which may be called the servoed Amble's method, was not programmed correctly, nor was good choice of the parameter λ estimated on any quantitative basis. The concept of the correction term $\lambda (O_d I + I_d O)$, which should be zero when $O = 1/I$, is that of servoing accomplished by integrators rather than operational integrators.

Alternative reciprocal calculations consisting of the Square and Integrate Method and Servoed Amble's Method are shown in Figures 9-3 and 9-4.

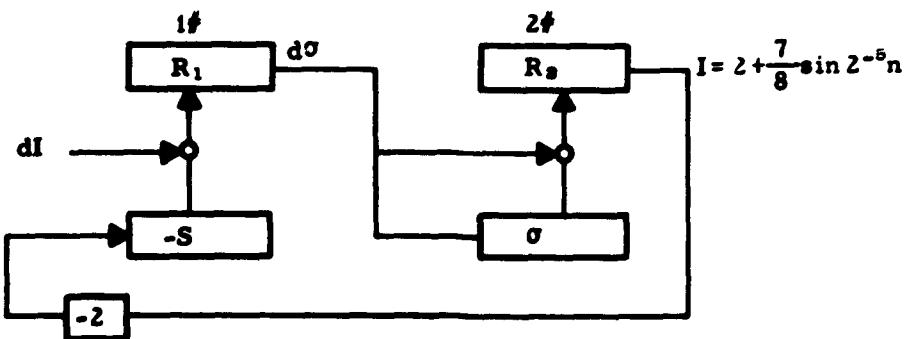


Figure 9-3. Alternative Reciprocal Calculations

Notation and
Differential Equations

$$\begin{cases} S = \theta^2, & = 1/1 \\ d\theta = -SdI \\ dS = 2\theta d\theta \end{cases}$$

Run Results: Mean absolute error in σ of 0.08 after 1800 iterations (about 1/6 the error rate in Amble's method).

Integrations Algorithms (Classically Mechanized)

$$\begin{cases} S_1 = 0.5, S_2 = 1/12 \\ S_1^2 = -0.5, S_2^2 = -1/12 \end{cases}$$

Initial Register Values

$$\begin{cases} y_1 = -0.25 \cdot 2^{-10} \\ y_2 = +0.50 \cdot 2^{-10} \end{cases}$$

$$\begin{cases} R_1 = 0.50 \cdot 2^{-10} \\ R_2 = 0.50 \cdot 2^{-11} \end{cases}$$

(2) Servoed Amble's Method

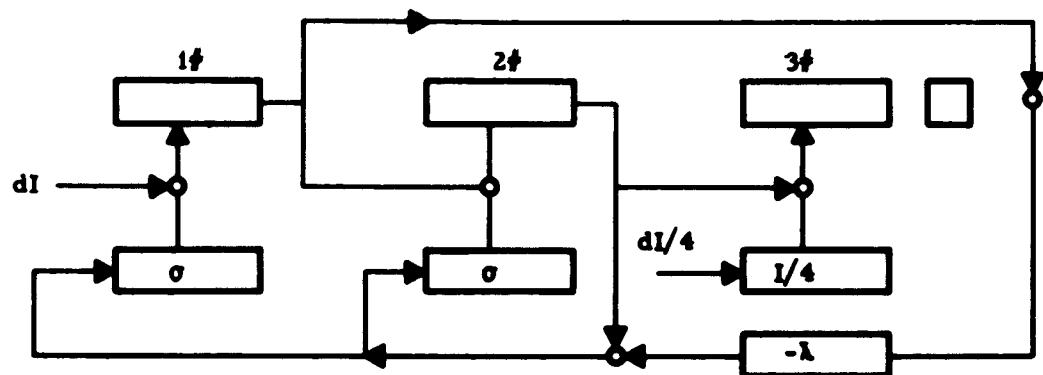


Figure 9-4. Servoed Amble's Method

Differential Equation:

$$d\sigma = -\sigma^3 dI - \lambda$$

(Linear Servo term $\sigma dI + I d\sigma$)

Integration

Algorithm

(Classically
Mechanized)

$$\begin{cases} S_1^1 = +1/2, S_0^1 = +5/12 \\ S_1^2 = +1/2, S_0^2 = +5/12 \\ S_3^2 = -1/2, S_3^3 = -1/12 \end{cases}$$

Initial Register :

Values

$$\lambda = 2^{-5}$$

$$\begin{cases} y_1 = 0.52292997 \\ y_0 = 0.50292997 \\ y_3 = 0.5 \\ R_1 = 0.5 \\ R_2 = 0.5 \\ R_3 = 0.5 \end{cases}$$

4. Reciprocal Calculation by Single Increment DDA with Elaborated Algorithm and Derivation of Digital Stieltjes Integration Algorithms -

a. Initial Efforts in Algorithm Development - In the initial runs of single increment DDA reciprocal calculation, an observed lack of response to first (and higher) order integration algorithm change in quantitative degree called for by the theory of numerical integration was interpreted to mean a lack of transmission to the output of the DDA integrator of first (and higher) order algorithm terms. Both DDA integrators in the reciprocal calculation by Amble's method are assigned functions of what may be called digital Stieltjes integration, the independent variables not being full rate. The lack of algorithm transmission and the observation regarding calculation structure characteristic of conditional transfer in Stieltjes integration served to assist the choice of direction of the detailed examination of the initial simulation runs. Error phenomena were highly magnified by the choice of high frequency inputs. It was observed that diaphantine phasing of Δy and Δx variables in an integration $y \Delta x$ led to highly sporadic action in the transmission of first order terms. While transmission may be sporadic, causing noise like errors, the average transmitted value would be expected, with proper digital algorithm, to correspond to the correct whole word values. The idea of reducing the degree of sporadic action by using a smoothed estimate of Δy was considered. A slowly changing smoothed value will

be transferred any time a Δx is non-zero. A smoothing calculation which is readily mechanized is

$$\Delta \tilde{y}_n = 2^{-3} \left[\Delta y_n - \Delta \tilde{y}_{n-1} \right] + \Delta \tilde{y}_{n-1} \quad (\text{IX-17})$$

The $\tilde{\Delta}y$ is used in the integration algorithm and is not used in the y register. The correct numerical integration algorithm (with $S_1 = 1/2$ and $S_2 = 5/12$) together with the first order term using $\tilde{\Delta}y$ was selected for simulation. The only difference in the simulated DDA from that of run (1) was the use of $\tilde{\Delta}y$. Simulation results indicated a modest improvement over the least poor run (Run 4) of the initial simulations, the magnitude of improvement being comparable to the improvement of Run 4 over Run 1. The lack of really a major improvement in using the $\tilde{\Delta}y$ was interpreted to imply that the random element of error was not the primary error source.

- b. Derivation of Digital Stieltjes Integration Algorithms - As a result of feedback (with one iteration delay) of outputs in Amble's method, this method when applied to a whole word incremental computer rather than DDA calls for the precise application of numerical Stieltjes integration algorithm (in essentially whole number incremental computation) in terms of virtual variables (refer to Chapter III) having the classical form,

$$\left[y_n + \frac{1}{2} \Delta y_n + \frac{5}{12} \Delta^2 y_n \right] \Delta x_n \quad (\text{IX-18})$$

for second order accuracy. All simulation evidence for single increment DDA indicated that the basic error source

of the DDA computation must stem from inappropriateness of the classical digital method of effecting the first (and higher) order algorithm terms in the conventional (or classical) manner of directly using the transmitted bit with half weight to represent $1/2 \Delta y_n$ in the algorithm. Since full rate integration ($\Delta x_n = \Delta t$) is successful with this direct representation for the first order term, the breakdown is associated with the fact that $\Delta x_n = 0$ for intervals in a pulse stream representation of Δx_n . Preliminary analysis (extended to include the complicating factor of feedback-induced-lag at a later date) implied that a major error results in algorithm realization by the conventional technique by not taking into account that the transferred quantity effected when $x \neq 0$ actually represents the integral increment over the period since last transfer. On this basis the proper algorithm for unlagged y variables to realize first order digital Stieltjes integration is

$$\left[y_n + \frac{1}{2} \sum_{p=n^*_{n+1}}^n \Delta y_p \right] \Delta x_n \quad (\text{IX-19})$$

where n^*_{n+1} is the iteration number of the preceding non-zero x_n , as seen by inspecting the schematic in Figure 9-5 and employing the trapezoidal rule.

In the reciprocal calculation, feedback of y induces a lag of one iteration which must be corrected for. The discussed digital Stieltjes algorithm viewpoint calls for introducing a lag (especially for low rate phases of Δx) which may be a large number of iterations, as in the example of the schematic being five iterations lag. Simulation evidence clearly showed

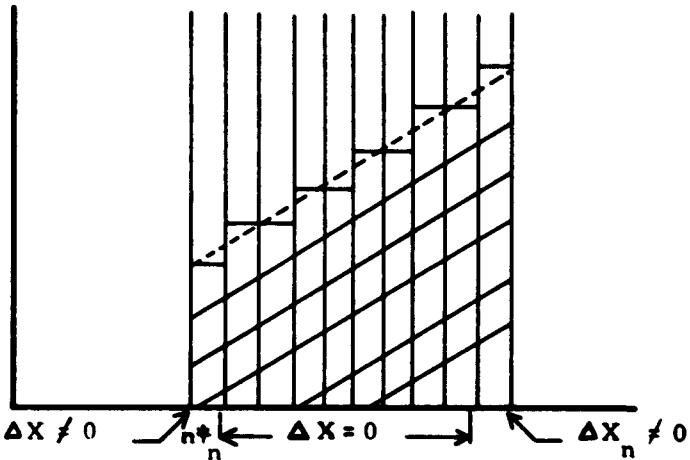


Figure 9-5. Iteration Number (n^*) Preceding Non-zero X_n

that further lag introduction would be deleterious (see runs (1) through (4)) in reciprocal calculation performance, which appeared to be contradictory to the principle of digital Stieltjes integration described. Resolution of the apparent contradiction was seen to lie in the detailed effects of the feedback of y delineated only by a precise statement of the purpose and method of digital representation, a degree of freedom in assignment that is at the choice of the designer. This statement and implications were not clearly resolved until a later date but are presented in parallel with preliminary efforts so that simulation results and theory may be correlated by the reader.

Analysis of the source of the Stieltjes transfer error effect is based on the difference between proper phasing of Δx used for y register update.

If update with Δx is assigned the role of bringing the value of x to the instantaneously correct value at a fixed time $\Delta \tau$ before, then if Δx , a single increment quantity, were programmed for transfer use rather than y update (i.e. at another integrator), an inconsistent time phase between transfer and y algorithm value is implied. This is because Δx calls for transfer, typically, after a number of iterations in which no transfer occurred, and it may be hypothesized that integrator outputs sent directly to a y register update exactly for $\delta \tau$ before. The integrator output necessarily represents the integral increment over the period since the last transfer displaced by $\delta \tau$. The proper first order algorithm for the integral increment therefore is based on the y value at the middle of the time interval since the last transfer displaced $\delta \tau$. In the reciprocal calculation the y register receives inputs delayed one iteration. For the case* $\delta \tau = 1/2 \tau$, mechanization and programming are based on the criterion of making resolution 1/2 the least bit. For the case $\delta \tau = 0$ the updated quantity represents the true variable at that instant. In either case chosen for design purposes, and adhered to consistently in design and programming, the value of $\delta \tau$ is less than one iteration interval. Digital Stieltjes integration algorithm for y undelayed (as by being fed Δy from a previous word time of the same cycle) is therefore based on the total y change since last transfer denoted.

* τ = iteration interval.

$$(\Sigma \Delta y)_n = \sum_p^n = \Delta y_{p,n^*} + 1 \quad (\text{IX-20})$$

where n^* is the iteration number of the last transfer (not the transfer of the present n^{th} iteration if any). In this case the second order digital Stieltjes integration algorithm for unlagged y variable is

$$\Delta I_n = [y_n - \frac{1}{2} (\Sigma \Delta y)_n - \frac{1}{2} \Delta (\Sigma \Delta y)_n] \Delta x_n \quad (\text{IX-21})$$

where y_n , x_n , are virtual variables. In the case of y variable, lagged as a result of computation of y during the last iteration cycle, a new algorithm is derived as follows: (1) The above stated digital Stieltjes integration formula is interpreted as being applicable with properly modified estimates of y_n , Δy_n , which take into account lagging effects. (2) The estimates of unlagged y , Δy must take into account the correlation of Δx and Δy changes which, for example, in reciprocal calculation, will be seen to be all important. Thus using numerical extrapolation formulae such as

$$(y_n) \text{ numerical lag corrected est.} = y_n \text{ lagged} + \Delta y_n + \Delta^2 y_n \quad (\text{IX-22})$$

may be completely inadequate if a correlation of the time of Δx and Δy changes exists, since the conventional algorithm implementation technique is tacitly based on averaging effects for a total ensemble rather than a correlated subset. To provide a basis for total ensemble averaging, make use of the statement that the function consisting of the arithmetic sum of Δy changes which occurs in period between $\Delta x \neq 0$ occurrences may be phased without sensitivity to correlated effects, rather reflecting the changes primarily of genuine interest in y function represented. Thus, if $\Delta y \neq 0$ occurred always

k iterations after $\Delta x \neq 0$, the sum of Δy over the Δx change cycle can be the same number at all phases relative to the iteration at $\Delta x \neq 0$, and represent a constant rate of a represented variable. The analogue of this criterion is filtering of an assigned frequency in a finite memory filter by averaging over the period. The generality with which effects of correlated times of Δx , Δy changes occur can be removed and is then indicated as the generality with which digital Stieltjes integration algorithms can be developed for various computation applications. In order to derive a lag correction algorithm from the algorithm stated for the unlagged Δy_n , Δx_n , there should ultimately be taken into account that the latter has its firmest basis where y is completely independent of x . In this case, the full ensemble averaging, which gives basis to the generalization of classical concepts of realization, firmly holds. Assume for the moment the abstract situation that the next iteration results at $n + 1$ are known at the n^{th} iteration. Then the lag correction algorithm based on the algorithm without y lag should ideally compute for approximate second order algorithm (where second difference with 1/12 coefficient is neglected)

$$\Delta I_n = [y_{n+1} - \frac{1}{2} \sum_{p=n+2}^{M+1} \Delta Y_p - \frac{1}{12} \Delta \sum_{p=n+2}^{n+1} \Delta Y_p] \Delta X_n \quad \text{neglected (IX-23)}$$

Ideal

Lag corrected

Algorithm

$$= [y_n + \frac{\Delta Y_{n+1}}{2} + \frac{\Delta Y_{n+2}}{2} - \frac{1}{2} \Delta \sum_{p=n+1}^n \Delta Y_p - \frac{1}{12} \Delta \sum_{p=n+2}^{n+1} \Delta Y_p] \Delta X_n \quad (\text{IX-24})$$

$$= [y_n + \Delta Y_{n+1} - \frac{1}{2} \sum_{p=n+1}^n \Delta Y_p + \frac{1}{2} \Delta (\sum_{p=n+2}^{n+1} \Delta Y_p)] \Delta X_n \quad (\text{IX-25})$$

steps two and three involving identities including

$$\Delta y_{n+1} = \Delta y_{n+1}^* + \Delta \left(\sum_{p=n^*}^{n+1} y_p \right) \quad (\text{IX-26})$$

and the $\Delta()$ operation n sum is meant for 1 iteration difference, not $(n-n^*)$ iterations.

The only unrealizable term is the last one (of the third step).

Employing the principle of filtering of effects with period $(n-n^*)$ iteration (where at n iteration a transfer is assumed to have occurred), the approximation

$$\Delta \left(\sum_{p=n^*}^{n+1} \Delta y_p \right) \approx \Delta \left(\sum_{p=n^*}^n \Delta y_p \right) \quad (\text{IX-27})$$

should hold in ensemble averaging to second order. In this case the lag correction digital Stieltjes integration algorithm for approximate second order accuracy is

$$\Delta I_n = \left[y_n + \Delta y_{n^*} - \frac{1}{2} \sum_{p=n^*}^n \Delta y_p + \frac{1}{2} \Delta \left(\sum_{p=n^*}^n \Delta y_p \right) \Delta x_n \right] \quad (\text{IX-28})$$

should hold to the accuracy level of the algorithm for unlagged y^1 . In evaluating the algorithm for unlagged y in the case where Δx , Δy pulse stream phase correlation exists, it should be commented that a certain degree of correlation would be presumed to

¹The second order term presents the same transmission problems by conventional representation of Δy_p as in full rate integration. A generalization of use of derivary terms developed for the latter can in principle overcome this problem, be effected by replacing $\Delta \left(\sum_{p=n^*}^n \Delta y_p \right)$ by $\sum_{p=n^*}^n D_n$ where D_n is derivary.

be part of the proper representation of the variables. If a phasing led to error, it might be proper to initialize the R register to shift phase to better digital representation. The most important applications appear to involve feedback with delay for which the lag correction digital Stieltjes derived should not introduce errors by virtue of the lag itself.

- c. Reciprocal Calculation Runs for Single Increment DDA with Digital Stieltjes Integration Algorithm - The principles of digital Stieltjes integration discussed in the preceding section were not fully reconciled at the times of the simulations of single increment DDA. It was felt initially that the quantity $\sum_{p=n^*}^{n+1} \Delta y_p$ played an important algorithm role

$$\sum_{p=n^*}^{n+1} \Delta y_p$$

in the digital Stieltjes integration. Empirically and analytically, it was determined that the error properties of conventional DDA could be cancelled by further increasing lag correction. The combination of these considerations led to simulations using the algorithm,

$$\left[y_n + \frac{1}{2} \Delta y_n + \frac{5}{12} \sum_{p=n^*}^{n+1} D_p \right] \Delta x_n \quad (\text{IX-29})$$

for the lagged y variables (D_n being the derivary). Note that Amble's method in a DDA implies that an input to each y register can occur one iteration after transfer and no other time and only when a transfer occurred in both integrators. Thus in Amble's method the algorithm stated above is exactly

$$\left[y_n + \frac{1}{2} \Delta y_{n^*} + \frac{5}{12} \sum_{p=n^*+1}^n D_p \right] \Delta x_n \quad (\text{IX-30})$$

Using $\sum_{p=n^*+1}^n \Delta y_p = \Delta y_{n^* n+1}$ again, note that the lag correction

digital Stieltjes integration algorithm derived in the preceding section
is

$$y_n + \frac{1}{2} \Delta y_{n^* n+1} + \frac{1}{2} \Delta \sum_{p=n^*+1}^n \Delta y_p \quad (\text{IX-31})$$

Using the identity (where $\Delta()$ operator on sum is a 1 iteration difference)

$$\Delta \left(\sum_{p=n^*+1}^n \Delta y_p \right) = \sum_{p=n^*+1}^n \Delta^2 y_p \quad (\text{IX-32})$$

and identifying $\Delta^2 y_p$ with D_p the derivary, it is seen that the algorithm simulated in reciprocal calculation is identical to the analytically derived lag correction digital Stieltjes algorithm (in which $\Delta^2 y$ is represented by D) to within approximately second order accuracy. Runs were made for the two different input functions, differing in degree of oscillatory component. The derivary terms were omitted in the third run. Detailed parameter values are presented on the next page, which also defines the smooth first difference run made previously. Results were excellent, being a fraction of an increment error over 2000 iterations using the digital Stieltjes integration algorithm. A measure of assurance that the results were not an accidental cancelling of error for a fortuitous calculation is indicated by the consistent performance for two input frequency characteristics. Longer runs would probably yield good results, as no error buildup was detected.

d. Final Simulation Runs of Reciprocal Calculations by Single Increment DDA with Non-Classical Algorithm.

(1) Smoothed First Difference Algorithm

$$y_0 = .502924997$$

$$R_0 = .5$$

$$R_0^* = .487306875 \quad \text{First Differ. Smooth: } \Delta(\tilde{\Delta}y_n) = 2^{-3} (\Delta y_n - \tilde{\Delta}y_{n-1})$$

$$\text{Input: } I = \frac{1}{2 + \frac{7}{8} \sin(2^{-5} n)} \quad (\text{IX-33})$$

$$\text{Algorithm: } \left[y_n + \frac{1}{2} \Delta \tilde{y}_n + \frac{5}{12} D \right] \Delta x_n \quad (\text{IX-34})$$

I generated by addition
of $\Delta(\frac{7}{8} \sin 2^{-5} n)$ to an R*
register apart from DDA
system

Results: Average Error of .033 (1 pulse)
after 900 iterations (thereafter
increasing small improvement
over algorithms

(2) Digital Stieltjes Integration Algorithm

Initialization same as (1)

$$\text{Input: } I = \frac{1}{2 + \frac{7}{8} \sin(2^{-6} n)} \quad (\text{IX-35})$$

$$\text{Algol: } \left[y_n + -y_n^* + 1 - \frac{1}{2} \sum_{p=n^*+1}^n \Delta y_p + \frac{5}{12} \sum_{p=n^*+1}^n D_p \right] \Delta x_n \quad (\text{IX-35})$$

Results: Average Absolute Error .011
(1/3 pulse) in 2000 iterations.
Probably good in a much longer
run

(3) Digital Stieltjes Integration Algorithm.

Initialization same as (1) Input: $I = \frac{1}{2 + \frac{7}{16} \sin(2^n)}$ (IX-40)

Algor and Results essentially
same as (2)

(4) Same as (3) with the Second Order Terms Omitted.

Initialization same as (3) Results: Essentially same as (3)

Conventional DDA algorithms with classical methods of realizing first order terms had previously led to large effective first order algorithm error. Thus a very large step in accuracy improvement was obtained in removing these effective first order algorithm errors in reciprocal calculation by use of the new lag correction digital Stieltjes integration algorithm. Since the run results were essentially that called for by perfect computation within the resolution in all of the three runs, the fact that the omission of the second order terms in the last run made no difference in results implies no general conclusion one way or another regarding 2nd order algorithm terms in general computation. In parallel with this simulation effort directed toward evaluating and optimizing single increment DDA, was a demanding schedule which included, as well as the preparation of strap-down computer evaluation tapes, the simulation evaluation of a revolutionary multi-increment DDA with second difference communication. Inherent rate handling capability and potential input processing capability of a multi-increment computer of acceptable cost was seen to offer the basis of design of a full aerospace mission incremental computer; the primary goal of the contract effort. Further desirable simulations were not made of single increment DDA with the new algorithm.

The ultimate value of a single increment DDA design capable of division and Stieltjes integration of air data in conventional airborne navigation systems (where associated accuracy requirements are not high) is very great. Doppler damping of improved accuracy would be possible without a special analogue computer. Previous failures of conventional DDA in such functions have been due not only to the single bit increment design feature, but very significantly in inadequate algorithm. A two bit increment DDA should be provided together with the new digital Stieltjes algorithm in adapted form. Such a DDA is entirely acceptable in cost for conventional airborne navigation tasks, and provides two levels of improvement in air data handling.

- D. Reciprocal Calculation by the Multi-Increment QDD^c A With Single Increment Communication - During the early intermediate period of Phase II study, the design of a multi-increment computer capable of division was analytically developed. The study was made on the basis of concepts of second difference output and single increment communication for band limited variables. Simulations of single increment DDA with elaborated algorithm had indicated that significant improvement, relative to the conventional DDA algorithm case in computations involving Stieltjes integration, had been developed. An analytical basis for the digital Stieltjes integration, including relatively sophisticated lag correction methods, had not been clearly defined at that time. Overall, the implication of

aerospace computer application studies, conducted in parallel with these efforts, demonstrated that a full aerospace mission computer system, in which the incremental computer assumed the major computation task, must have a significantly higher computation capability than a single increment DDA could possibly attain. This conclusion was based on the assignment of such tasks as thrust cut-off, strap-down computations, and air data computations of re-entry, as sub-routines. All such computations are individually possible in a set of single increment DDA computers of very high iteration rate, tied into a single system. This is an expensive mechanization approach for the computation capability obtained.

Granting that input processing requires multi-increment computation, the mechanization advantages of time shared internal computations imply that a degree of multi-increment bit length for the latter less demanding computations is potentially available, and is called for in optimized design. A several bit increment internal computation not only removed basic elements of marginality characterizing single increment DDA, but is basically available in the time shared design of a computer which executes input processing. The benefits of quotient algorithm had been attained in efforts in the computer field in only single increment computers, such as variable single increment. Benefits were analytically shown possible with multi-increment mechanization in a highly integrated system. The object of simulations was to evaluate the quantitative performance of a mechanization derived

on the basis of the newly developed concepts of multi-increment computation, with second difference communication for a computation involving the process of division by the basic unit developed. For nondivision operations the structure of the mechanization insures the accuracy of a multi-increment DDA. A major goal of the simulation effort was to demonstrate the feasibility of multi-increment computation with second difference output and communication in a concrete example of computation.

1. Programming the Basic Processing Unit (QDPU) of the QDD²A with Approximate Second Order Algorithm - The basic transfer action of the QDD²A is a generalization of the conventional DDA, as several transfers to a single R register rather than a single transfer are performed in one cycle of operation. The basic integration algorithm operation effected in the conventional DDA, by modifying y register quantities in accordance with past values before transfer to the R register, is retained in form, and considerable quantitative correspondence exists in the QDD²A transfers. One principle general difference in internal operation, apart from output generation, when viewed as a purely serial process, is that a single R register is used instead of two (or three) R registers as in a conventional DDA. Thus the more nearly conventional DDA computer program used in preceding simulations could be modified with respect to internal operation to simulate a QDD²A program by programming changes which include the collection of transferred quantities stored temporarily in their separate R registers (modified so as to have no overflow action) and then placing the resultant in the R register of the last integrator associated with the R register of the QDPU. The

analytically derived most elementary unit of the multi-increment QDPU involves collection of two multi-transfers and a single transfer (the latter serving to effect an algorithm refinement for high accuracy). Thus the transfer action is that of two multi-increment DDA integrators and one single increment DDA integrator. A single multi-transfer unit and one single transfer unit could execute the cycle of operation for reciprocal calculation with the same cycle time (2 word time) as in a conventionally conceived serial multi-increment DDA provided with multi-increment communication. However, to effect the identical processing as the QDPU, there would have to be whole word communication of R register quantities. A second major difference in this QDPU operation is the output criterion (for second difference outputs) which is not of the natural overflow type. Hence the output criterion required a special programming modification to replace the simulation of natural overflow. The third basic processing modification is that in the absorption of inputs. The input second differences are accumulated in separate (short) registers to form first differences which are then used as Δx , Δy quantities in a conventional DDA integrator where the update $y + \Delta y$ and transfer $\tilde{y} \Delta x$ are executed. The nature of the simulation program for conventional DDA makes this direct. A final minor programming change is the replacement of second order terms of the derivary type with the direct second order differences communicated. It is clear that one of the many merits of second difference communication is that second order algorithm terms can be effected as simply as first order algorithm terms in a conventional DDA with first difference communication.

A programming of the elementary unit of the QDPU may be expressed generally as having the form:

$$(1) \quad \Delta R_n^{(3)} = \tilde{y}_n^{(3)} \Delta x_n^{(3)} + \Delta R_n^{(2)} + \Delta R_n^{(1)} \quad (\text{IX-38})$$

$$\Delta R_n^{(2)} = \tilde{y}_n^{(2)} \Delta x_n^{(2)}$$

$$\Delta R_n^{(1)} = \tilde{y}_n^{(1)} \Delta x_n^{(1)}$$

where $\Delta R_n^{(k)} = R_n^{(k)} - R_{n-1}^{(k)}$, $R_n^{(k)}$ being contents of the k^{th} R register,

$$\tilde{y}_n^{(k)} = \left[y_n^{(k)} + S_1^{(k)} \cdot \Delta y_n^{(k)} + S_2^{(k)} \cdot \Delta^2 y_n^{(k)} \right] \begin{array}{l} \text{Classical Int-} \\ \text{egration Algo-} \\ \text{rithm Realization} \end{array} \quad (\text{IX-39})$$

or

$$y_n^{(k)} = \left[y_n^{(k)} + S_1^{(k)} \sum_{P=n_n^*+1}^n \Delta y_p^{(k)} + S_2^{(k)} \sum_{P=n_n^*+1}^n \Delta^2 y_p^{(k)} \right. \\ \left. + \Delta y_{n_n^*+1}^{(k)} \right] \begin{array}{l} \text{Digital Stieltjes} \\ \text{Integration Algorithm} \end{array} \quad (\text{IX-40})$$

where n_n^* is the iteration at which the previous non-zero transfer occurred.

$$(2) \quad \Delta^2 \theta_n^{(3k)} = \text{sgn} R_n^{(3k)} \text{sgn} (\tilde{y}_n^{(2)}) u(2|R_n^{(3)}| - K|\tilde{y}_n^{(2)}|) \quad (\text{IX-41})$$

where $\Delta^2 \theta_n^{(3k)}$ is the k^{th} QDPU output and

$$(3) \quad \Delta x_n^{(r)} = \Delta x_{n-1}^{(r)} + \Delta^2 y_n^{(3k)} \quad (\text{IX-42})$$

$$\Delta y_n^{(s)} = \Delta y_{n-1}^{(s)} + \Delta^2 \theta_n^{(3k)}$$

are update rules when the k^{th} QDPU output is programmed to be input to the r^{th} integrator for x or s^{th} integrator for y , or for external inputs.

$$\Delta x_n^{(r)} = \Delta I_n^{(m)} \quad (\text{IX-43})$$

$$\Delta y_n^{(s)} = \Delta I_n^{(m)} \quad (\text{IX-44})$$

where external input $\Delta I_n^{(m)}$ enters QDPU.

2. Program for Reciprocal Calculation by the QDPU with Approximate Second Order Algorithm - In Chapter II a formal approximate second order numerical algorithm for division processes by integration is derived. The computation of the integral increment

$$\Delta \theta_n = \frac{\int_{(n-1)\tau}^{n\tau} \frac{P}{V} dx}{\tau} \quad (\text{IX-45})$$

by second difference output is shown to involve the R register computation

$$\Delta R_n = \tilde{p}_n \Delta x_n - \tilde{v}_n \Delta \theta_n^D - \left[v_{n-1} + \frac{\Delta v_n}{12} \right] \Delta^2 \theta_n^D \quad (\text{IX-46})$$

where \tilde{p}_n , \tilde{v}_n are modified p_n , v_n quantities for integration algorithm appropriate for lagged or unlagged quantities. The reciprocal calculation

$$\theta = 1/I$$

satisfies the differential equation

$$d\theta = -\frac{\theta dI}{I}$$

hence the integral increment of $\Delta \theta_n$ is

$$\Delta \theta_n = \int_{(n-1)\tau}^{n\tau} \frac{[-\theta] dI}{I} \quad (\text{IX-47})$$

which has the general form of the algorithm analysis, taking $p = -\theta$, $q = I$, $x = I$. Here V , x are unlagged since inputs ΔI and updating of I are chosen as unlagged. Since θ is a feedback quantity the quantity p is lagged. Therefore the appropriate integration algorithms are

$$\tilde{P}_n = - \left[\theta_n + \frac{1}{2} \Delta \theta_n + \frac{5}{12} \Delta^2 \theta_n \right] \quad (\text{IX-48})$$

$$\tilde{V}_n = \left[I_n - \frac{1}{2} \Delta I_n - \frac{1}{12} \Delta^2 I_n \right] \quad (\text{IX-49})$$

$$V_{n-1} + \frac{\Delta V_n}{12} = \left[I_n - \frac{17}{12} \Delta I_n \right] \quad (\text{IX-50})$$

the last relation entering as a first order accuracy term adequate to obtain the approximate second order accuracy. The analysis of Chapter II therefore calls for in reciprocal calculation the programming according to the form stated in the last section such that

$$\begin{aligned} \Delta R_1 &= \left[\theta_n + \frac{1}{2} \Delta \theta_n + \frac{5}{12} \Delta^2 \theta_n \right] \left[-\Delta I_n \right] \\ \Delta R_2 &= \left[I_n - \frac{1}{2} \Delta I_n - \frac{1}{12} \Delta^2 I_n \right] \left[-\Delta \theta_n \right] \\ \Delta R_3 &= \left[I_n - \frac{17}{12} \Delta I_n \right] \left[-\Delta^2 \theta_n \right] \end{aligned} \quad (\text{IX-51})$$

where the superscripts D are dropped from θ quantities which are understood to be inputs rather than outputs. The output criterion derived in Chapter VII for second difference output is

$$\Delta^2 \theta_n^{(1)} = \operatorname{sgn} R_n^{(3)} \operatorname{sgn} \tilde{V}_n U \left(2 |R_n^{(3)}| - K |\tilde{V}_n| \right) \quad (\text{IX-52})$$

where

$$R_n^{(3)} = R_{n-1}^{(3)} + \Delta R_n^{(1)} + \Delta R_n^{(2)} + \Delta R_n^{(3)}$$

and \tilde{V}_n is stated above and K selected for given scale of second difference output.

3. Scaling and Input Generation for the QDPU in Reciprocal Calculation Simulations - The input ΔI was generated in the same programming structure as in single increment DDA simulations but with a quantization for multi-increment rather than single increment. The calculations simulated were $\theta = 1/I$ where

$$I = A + B \sin \theta_0 n$$

for the cases $A = 2$, $B = 7/8$ and $A = 3$, $B = 7/8$ where generally $\theta_0 = 2^{-5}$. For a QDPU with a single multi-transfer unit that cycles in two word times, the first simulation corresponds to the same physical input tested in single increment DDA (about 1 cps input for 200 iter/sec DDA). For the ODD³A with two multi-transfer units for one channel of computation, the simulation evaluates results for inputs with twice the frequency tested for conventional DDA. The program for generation of ΔI inputs to the QDPU is essentially exterior to the ODDA program structure. Multi-increment ΔI were generated by R register action as in a DDA with multi-increment output, in which whole word ΔI increments are added to R with a bias-free multi-increment output criterion being programmed. The inputs are expected to closely simulate the accumulated inputs of a pulse stream input

device, where accumulation occurs over the iteration interval of the QDDA. The maximum increment ΔI of physical input I is $< 2^{-5}$ part of full scale of I , for both single and multi-increment computation. As a result of the fact that the second difference changes at most 2^{-10} , it was possible to simulate ODDA with single increment second difference communication in a computation with 10 bit registers instead of the short (5 bit) registers forced on a single transfer DDA. The computed input was actually $\theta_o^{-1} \Delta I$ computed as five bit numbers ≤ 1 in absolute value therefore having scale of 2^{+5} . The computed ΔI was programmed as independent variable with unit scale and as y register update \bar{I} with scale $2^{-2} \times 2^{-5} = 2^{-7}$, the first factor taking into account $\bar{I} = 2^{-2} I$ and the second taking into account physical scale; then $\bar{I} < 1$. The $\Delta^2 \theta$ outputs of +1, -1 or 0 were programmed to update θ in a y register with scale 2^{-10} ; then $|\bar{\theta}| < 1$. The computation in machine variables was programmed with a machine scale so that

$$\begin{aligned} R_1 + R_2 + R_3 &= \bar{I}_n \left[-\bar{I}_n \right] \\ &+ \bar{I}_n^{(2)} \left[-S_{\Delta \theta} \cdot \Delta \theta_n \right] \\ &+ \bar{I}_n^{(3)} \left[-S_{\Delta \theta}^2 \Delta^2 \theta_n \right] \end{aligned} \quad (\text{IX-53})$$

has ΔR_i with 2^5 physical scale of $\Delta \bar{I}_n$ and since \bar{I} had physical scale, the scales $S_{\Delta \theta}$, $S_{\Delta \theta}^2$ were necessarily $S_{\Delta \theta} = S_{\Delta \theta}^2 = 2^{-3}$ since $\bar{I} = 2^{-2} I$ and $\Delta^2 \theta$ has 2^{+10} physical scale, as follows from the equation $2^5 = 2^{-2} S_{\Delta \theta} 2^{+10}$. The output computation utilized the same scale, hence $K = S_{\Delta \theta}^2$.

4. Initial ODPU Simulations - The initial simulation was primarily intended to establish the analytically derived principle of multi-increment computation with single increment output and communication. After program debugging was complete a successful run with algorithm terms (classically mechanized 1st and 2nd order terms), and input

$$S_1^{(1)} = \frac{1}{2}, S_2^{(1)} = \frac{5}{12} \quad (\text{IX-54})$$

$$S_1^{(2)} = \frac{1}{2}, S_2^{(2)} = -\frac{1}{12} \quad I = 2 + \frac{7}{8} \sin(2^{-5} n)$$

$$S_1^{(3)} = -\frac{13}{12}, S_2^{(3)} = 0$$

was executed in a 10,000 iteration run. A peak error* during the run was 0.0073 (absolute average over 50 iteration) in the computation involving increments < 0.0325 per iteration. Regarded as linear error growths, the improvement over conventional DDA was a factor of 150 for serial transfer ODPU. The contemplated CDPU would handle twice the frequency input with the same performance. Note that a second order term produced by the 1st order \bar{I} term of $\Delta R^{(3)}$ differs somewhat from the analytically derived algorithm.

The results would presumably have been improved somewhat, had the intended algorithm been simulated. Second order terms $\Delta R^{(1)}$ and $\Delta R^{(2)}$ are shown in later simulations to not significantly affect reciprocal calculation. The second run,

*Peak error in direct printout each 1000 iterations was 0.0047. The average absolute error printout is questionable here and in third run.

which will be described indicates that there may be greater sensitivity in the $\Delta R^{(3)}$ to terms of this order of magnitude. The second run was primarily intended to evaluate the importance of the ΔR_3 term, the approximate significance of which is deduced by the first order approximation.

$$\Delta R_2 + \Delta R_3 = -\left(\bar{I}_n - \frac{1}{2} \Delta \bar{I}_n\right) \left(\Delta^r_n + \Delta^2_n\right) \quad (\text{IX-55})$$

differing from ΔR_2 by use of $(\Delta^r_n + \Delta^2_n)$ instead of Δ^r_n .

Thus ΔR_3 primarily extrapolates $\Delta \theta$ as an independent variable iteration ahead. The second simulation program differed from the first only in that $\Delta R_3 = 0$ (discard of ΔR_3 term). In 1000 iterations the run with modification for $\Delta R_3 = 0$ showed an error of 0.030, and thereafter degraded to near full magnitude error in 3000 iterations. Since ΔR_3 is a term of 1st order, the omission of ΔR_3 would be expected to produce errors up to the magnitude comparable to 1st order algorithm error in a sinusoid calculation, as was found to be the case in reciprocal calculation. A third simulation had the purpose of testing a refinement in algorithm digital realization based on generalization of the digital Stieltjes algorithm technique derived during phase 2 first for single increment computation as then formulated. Conventionally mechanized integration algorithm was seen to fail in algorithm transmission for the cumulative period during which $x = 0$ till $\Delta x \neq 0$, which is most pronounced in single increment computation because of the high frequency of such periods. A unified algorithm technique for general DDA computation would incorporate this special action during the less frequent

periods in which $\Delta x = 0$ to $\Delta x \neq 0$. It will be noted that Ambie's method with multi-increment quotient algorithm computation would be expected to retain, in high degree, the specific property that outputs be generated only when $\Delta x \neq 0$, and further that, when generated, are delayed one iteration in feedback. Thus the specific format programmed according to the then formulated digital Stieltjes algorithm, is equivalent to the more general formulation, including lag correction digital Stieltjes integration algorithm derived in a preceding section. The general order of magnitude of effect expected by the refinement in multi-increment computation would be a first order algorithm error effect reduced by a factor of 2^{-2M+2} , where M is multi-increment bit length, according to the following argument which holds for only the zero crossings of the desired variable (not the partial rate of zeros in fraction representation where a large value is represented by the pulse stream). The frequency of zero crossings of the (desired) numerical variables is, of course, independent of M. The duration of $\Delta I = 0$ in digital representation is proportional to 2^{-M+1} . The 1st order integration algorithm integrand term magnitude is then proportional to 2^{-M+1} . The scale of effect in transfer $y \propto x$ is the product of these scales, hence the factor 2^{-2M+2} . The evaluation of partial rate effects for large Δx in single increment computation is more involved. For M = 5, as in the CDPU simulation the reduction of effect relative single increment would be 1/256, assuming the major error effect occurs around zero crossings of the desired variable. In the pertinent calculation the error effects is a 2×10^{-4} error

increment per iteration for single increment and accordingly is estimated as 0.8×10^{-6} for 5 bit increment computation. In 10,000 iterations an error of 0.008 would be consistent with this error model for digital Stieltjes integration effects, in agreement with the error level determined in simulation of the QDPU without the digital Stieltjes algorithm. The third simulation incorporated the special digital Stieltjes algorithm refinement just analyzed. A problem was presented in evaluation of results because of a probable error in programming the sum of absolute errors over 50 iteration intervals. The largest error in direct printout of a iteration result was 0.0047, recorded once each 1000 iterations. The single sample errors showed essentially linear amplitude growth. However, from the run start the printout of average absolute error for 50 iteration ensembles was a 0.0078 near start, growing to 0.0100. The latter appears to reflect a programming error with a bias in a modified average error evaluation routine. Assuming this to be the case, the digital Stieltjes algorithm refinement reduced error somewhat (from 0.0057 to 0.0047) in the multi-increment computation, but not in the degree hoped for. Perfect computation to within resolution could yield ± 0.0005 .

5. Simulations of Reciprocal Computation by the Multi-increment QDPU With Second Order Algorithm and Modified Input Functions - Preparation of a set of runs, simulating the QDD²A alternative second order algorithms and two different input functions, was carried out in a parallel effort with that for the first CDPU runs. Second order algorithms were

programmed to correspond to a conventional realization of individual algorithm terms but not also generalized digital Stieltjes algorithm realization for multi-increment computation, as in the preceding runs, which had not been fully evaluated (as a result of a programming error in the average absolute error estimate for ensembles of 50). These runs served to evaluate the degree of algorithm mechanization simplification tolerable for reciprocal calculation as well as performance improvement for a calculation with input, which more closely matches the demands of typical application i.e., a less demanding calculation. The first of this set of runs (the 4th of ODPU runs) evaluated the approximate second order algorithm, stated in an earlier section, and derived on the basis of theory developed in Chapter II, that is to within the level of approximate second order algorithm stated. The simulated algorithm is defined by

$$S_1^{(1)} = 1/2 \quad S_2^{(1)} = 5/12 \quad (\text{IX-56})$$

$$S_1^{(2)} = -1/2 \quad S_2^{(2)} = 0 \quad \text{Run 4(a), (b)}$$

$$S_1^{(3)} = -3/2 \quad S_2^{(3)} = 0$$

in $\Delta R_n^{(1)}$, $\Delta R_n^{(2)}$, $\Delta R_n^{(3)}$ respectively, differing from the derived algorithm by 6 percent of a second order term (in $\Delta R_n^{(3)}$). The labeling of runs 4(a) and (b) means runs for the calculations.

$$\theta = \frac{1}{\left(2 + \frac{7}{8} \sin 2^{-5} n\right)} \quad (\text{a}) \quad (\text{IX-57})$$

$$= \frac{1}{(3 + \frac{7}{8} \sin 2^{-5} \frac{n}{n})} \quad (b) \quad (IX-58)$$

respectively. The first calculation (a) involves Δ register swinging from $\frac{8}{23}$ to $\frac{8}{9}$ in the same time in (b) the Δ register swings from $\frac{8}{17}$ to $\frac{8}{31}$ corresponding to 2.5 times the average rate of change. Runs for comparison with runs 4(a), (b) were

$$\begin{array}{ll} s_1^{(1)} = 1/2 & s_2^{(1)} = 0 \\ s_1^{(2)} = -1/2 & s_2^{(2)} = 0 \\ s_1^{(3)} = -1/2 & s_2^{(3)} = 0 \end{array} \quad (IX-59)$$

Runs 5(a), (b)

and

$$\begin{array}{ll} s_1^{(1)} = 1/2 & s_2^{(1)} = 1/2 \\ s_1^{(2)} = -1/2 & s_2^{(2)} = -1/16 \\ s_1^{(3)} = -1 & s_2^{(3)} = 0 \end{array} \quad (IX-60)$$

Runs 6(a), (b)

Before presenting results of these simulations, it is desirable to indicate that the general algorithm form has equivalences when 1st and 2nd order terms are mechanized in the conventional manner. One of these equivalences involves ΔR_1 and ΔR_3 such that since $\Delta^2 \theta_n \Delta I_n = \Delta I_n \Delta^2 \theta_n$ it follows that any choice of $s_2^{(1)}$ and $s_1^{(3)}$ such that $(s_2^{(1)} + s_1^{(3)})$ is unchanged implies algorithm equivalence (in convention realizations).

Thus, for example Run 4 with $s_2^{(1)} = 5/12$, $s_1^{(3)} = -3/2$ is equivalent to a run with $s_2^{(1)} = 0$, $s_1^{(3)} = -\frac{13}{12}$.

For Run 5 with $S_2^{(1)} = 0$, $S_1^{(3)} = -1/2$ is equivalent to a run with $S_2^{(1)} = 1/2$, $S_1^{(3)} = -1$. For Run 6 with $S_2^{(1)} = 1/2$ and $S_1^{(3)} = -1$ is equivalent to a run with $S_2^{(1)} = 9/16$, $S_1^{(3)} = -17/16$. As stated first, Run 5 simulates a QDPU with two multi-transfers, one of which involves $(\frac{R_1}{n} + \frac{R_2}{n})$ as independant variable (by combining R_2 and R_3). Simulated ODD²A and performance which agreed with the initial runs in that greatest error occurs at the peak of α values and attenuates after peaks with only a gradual error drift being generated permanently. All runs for calculation (a) had errors within a factor of two of each other at the same iteration count. Near the peak α values, Runs (4), (5), (6) (a) led to peak errors of 0.010, except for Run 5(a), which was printed out at one point closest to the maximum α for which error of -0.0180 at $\alpha = 0.89$, and error of 0.0078 at $\alpha = 0.84$, were observed near the end of 10,000 iteration run. It is deduced that more detailed printout would have showed that average error over a α cycle would be substantially under the peak errors at the α peaks of 1/4 increment (peak) error (3 bits of a 5 bit increment), perhaps being 1/8 increment average error (2 bits of 5 bits increment). The less demanding calculation, labeled (b), showed for Runs (4), (5), (6) (b) a similar closeness in comparative performance throughout run periods, where essentially linear error growth was observed. As expected, markedly better absolute performance resulted, error drift (from initial biases) building up to about -0.003

in each case, amounting to about 10 percent of an increment per iteration after 10,000 iterations. The comparative 40 percent average rate of variation of the I function in calculation (b) compared to (a) was more than matched by fractional improvement. The implications of simulation results which showed closely similar performance for different second order algorithms (mechanized by conventional identification of communicated increments with increments of the true variable, rather than use of digital Stieltjes identification, meaning that substantial improvement in performance probably must be attained by the refinement of digital Stieltjes algorithm. This conclusion is considered all the more pertinent for three bit increment computation instead of the five bit increment computation simulated. A seventh run was made with slight perturbations in initial condition of I_0 and θ_0 relative Run 4(a) to determine the degree of algorithm transmission improvement by subsignificant biases (longer register length with artificial word length). Thus Run 7(a) used

$$I_0 = 1/2 + 3 \times 2^{-15} \quad (\text{IX-61})$$

$$\theta_0 = 1/2 - 3 \times 2^{-15} \quad (\text{IX-62})$$

as a modification of Run 4(a). Run 7(a) was similar to Run 4(a) but had peak error of 0.0087 instead of 0.0100, amounting to a 13 percent improvement. The limited level of improvement indicated that the digital Stieltjes refinement was still the direction for any major improvement.

In summary of overall QDD²A performance as compared to conventional DDA, the error growth in reciprocal calculation of 10^{-6} per iteration displayed in five bit increment QDD²A simulations of calculation (a) as compared to 2×10^{-4} per iteration for conventional DDA, demonstrates a factor of 200 improvement in accuracy. The potential in this case is perhaps 1000 with digital Stieltjes algorithm refinement. The stated comparison has been made for a two word time QDPU with a single multi-transfer unit. A parallel QDPU (one word per cycle) would be able to compute reciprocal for inputs with twice the frequency simulated with the same performance.

- E. Simulations of Full Rate and Variable Rate Sinusoid Calculations by Single Increment DDA with Elaborated Algorithm and the QDD²A -**
The simulations during Phase II of full rate and variable rate sinusoid calculations by various DDA mechanizations had the objectives of:
1. Duplicating certain runs made during Phase I on the Alwac computer in order to checkout programming during Phase II for the 704 computer, and verify previous results on Alwac during Phase I. The duplicated runs were full rate sinusoid calculations. Evaluation of individual design features of the derivative integrator of Phase I was desired.
 2. Evaluation of error magnitudes and error correction in sinusoid calculation by conventional and elaborated DDA mechanizations where the independent variable is partial or variable rate. Specifically to evaluate the digital Stieltjes integration

algorithm developed during Phase II as a result of reciprocal calculation simulations.

3. Provided that the programming schedule for strap-down processor evaluation permitted, to evaluate the QDD²A in multi-increment sinusoid calculation; however this did not prove possible. There was preliminary consideration of QDPU single increment mode for sinusoid, but the system application of the QDD²A for sinusoid generation was determined to be best chosen for execution in multi-increment computation as a result of sinusoid error sensitivity in relation to processing rate, resolution, and general precision, which is attained completely within the mechanization structure of the proposed QDPU developed on general computation function bases.
 - a. Simulation of full rate sinusoid calculation by conventional and derivary DDA. Results of the full rate single increment sinusoid calculation simulations provided verification of Phase I quantitative and comparative performance of conventional DDA and the DDA with derivary communication developed during Phase I. Three innovations characterized the derivary integrator:
 - (1) The technique of subsignificant biases of y registers for improved algorithm transmission, at the price of increased register length with given granulatiry.
 - (2) The communication of second order terms in a digital representation in which algorithm transmission is improved. Derivary and programmable algorithm features in otherwise conventional DDA were

simulated. Derivary uses the changes in the transferred quantity to the R register of communicating integrator as involved in Phase I.

- (3) Overflow inhibitor technique which greatly reduces roundoff error effects associated with low rate integrator output (as involved in Phase I).

The reported performance improvement was verified iteration for iteration, but the relative improvement accordable to the derivary innovation was not fully determined. The value of the overflow inhibitor was verified repeatedly in these simulations. The value of subsignificant biases was confirmed but a question of the essentiality of the derivary terms was resolved to a question of programming error in a late phase run. The case in point was a run (made serially with two other runs at the same 704 run appointment) which was supposed to simulate the derivary integrator with an overflow inhibitor and subsignificant biases in which the derivary communication was cut off. The run duplicated iteration for iteration the results of the complete derivary integrator repeated in Phase I and II. While for the high frequency, sinusoid simulated this is not necessarily impossible as a result of chance diaphantine relations of R register value for the sinusoid amplitude, a programming error of failing to cut off the derivary terms is judged likely. The analytical implication that subsignificant biases primarily offer the improved transmission of second order terms rather than the first

order terms, which on analytical bases should be transmitted, tends to indicate a programming error was made. The specific simulation parameters and summarized results (the latter discussed above) are presented below:

Run 1 Repeat of run summarized on page 236 of the Phase I report. Run results identical on 704 at each iteration. Again the large reduction of frequency shift from the theoretical relative conventional DDA.

Run 2 Repeat of run summarized on page 232 of the Phase I report. Results same on 704.

Run 3 Simulated DDA to repeat Run 1 identically except for cutoff of derivary terms. Results identical to Run 1 iteration for iteration indicating possible programming error.

The design implications for full rate integration by near conventional integrators, if the results of Run 3 were correct, would be a DDA integrator with an overflow inhibitor and subsignificant bias developed during Phase I but without derivary communication. Apparently, register length increase could be avoided by introducing a bias of 1/2 bit (rather than smaller biases) in the y register, since it is analytically observed that the only palpable purpose of the bias is to assist transmission of the algorithm terms of least magnitude which in first order algorithm is 1/2 bit (whereas in derivary second order algorithm it is much smaller). Other simulations during Phase II of DDA and

QDD²A do not clearly resolve the question of second order integration since digital Stieltjes integration effects in partial rate computations involve errors of magnitude lying between first and second order effects. However, in all cases, the use of second order terms either improved results or made insignificant difference in results (being masked by the large error effects alluded to). Recommendations are that further runs be made to resolve the question of relative significance of these component design features (especially for the generalized forms of these features in the important problem of multi-increment computer design).

- b. **Uncompleted Programming of QDD²A Sinusoid Calculation -** Programming assignments were made for simulation of multi-increment sinusoid calculation by the QDDA in the event that the simulations could be successfully completed before previously scheduled programming efforts for strap-down processor evaluation. A first program was laid out but was not successfully debugged because adequate programmer time was not available. The intent of the QDPU sinusoid runs was to establish the level of accuracy in multi-increment computation with and without overflow inhibitor action designed to be a generalization of the overflow inhibitor technique developed in Phase I. Since QDPU reciprocal calculation does not involve registers with null values, the play of error factors for which the overflow inhibitor was developed to overcome does not occur in reciprocal calculation.

The overflow inhibitor mechanization design will require certain modifications to be a generalization for multi-increment (from the single increment case). The considerations in the development of the generalization are similar to those in the generalization of digital Stieltjes algorithm for single increment to multi-increment computation. The first runs contemplated for overflow inhibitor test were based on inhibition when the least significant bit of y register is zero. The general output criterion is modified to inhibit output until the condition no longer exists. In computing a full rate sinusoid in a QDD³A it is possible to double register length (M to 2M) of the sinusoid relative a single increment DDA yet retain single increment communication. The mechanization price of multi-transfer operation ranges from that of conventional M bit multiplier to that of a three bit increment multiplier in the D² multiplier developed in Chapter XIII, which is capable in this case of a remarkable mechanization economy when M > 3 (and second differences are single increment). Contemplations were to first simulate the sinusoid for M = 5 case for comparison with reciprocal calculation results which had M = 5.

- c. Simulations of Partial and Variable Rate Sinusoid Computations by Single Increment DDA and Further Verification of the New Digital Stieltjes Integration Algorithm.
The computation of sin I, cos I where the input is formed by R register overflow with transfers of $\Delta I = K_1 + K_2 \Delta(\sin \theta_0 n)$, was chosen for a study of partial and variable rate

sinusoid computations of single increment DDA with overflow inhibitor with and without digital Stieltjes algorithm. The first two runs simulated a sinusoid calculation with constant partial rate input $K_1 = 5/16$, $K_2 = 0$ with four bit registers for sine and cosine with added subsignificant bits and initialization and algorithm given by:

$$y_o = .0078125 \quad y_o = .37890625 \quad \text{Initialization}$$

$$R_o = .50390625 \quad R_o = .4921875$$

$$S_1^1 = .5 \quad S_1^2 = -.5 \quad \text{Integration Algorithm}$$

$$S_0^1 = .4167 \quad S_0^2 = -.08333$$

$$T = 2^{-4} \quad \text{Overflow Inhibitor Magn. Criterion}$$

Simulations were made for the two cases of not using and of using the digital Stieltjes algorithm derived earlier in this chapter. The first case simulates the derivative integrator developed (during Phase I) for full rate calculations, while the latter simulates the new digital Stieltjes integrator developed for general independent variables. Results in the two cases are expressed as peak average absolute error over ensembles of 50 iterations of the larger of the computed sines and cosines error values, summarized as follows:

Partial (Constant) Rate Simulations Results

Period:	4000 iterations	8000 iterations
Error of Derivary Integrator System	.082	(Run stopped)
Error of Digital Stieltjes Integrators System	.020	.022

Pulse size was .0625 and exact stable calculation within resolution would result in error $1/4 (.0625) = .0156$ error in the average absolute error. These results are another confirmation of the value of the digital Stieltjes algorithm for independent variables which are not full rate.

A second set of simulations was programmed and run for the purpose of evaluating the relative performance of conventional and digital Stieltjes algorithm single increment DDA for inputs characterized by modulated partial rates. Programming errors were detected on the basis of the exact solution used to evaluate the runs which did not correspond to assigned solution; therefore no useful results are available. The assigned runs were for the same initial DDA conditions as the preceding runs for pure partial rate modified for the new inputs:

$$\Delta I_n = 2^{-4} \quad (IX-63)$$

$$\Delta I_n = 2^{-4} + 7/8 \Delta (\sin 2^{-n}) \quad (IX-64)$$

$$\Delta I_n = 2^{14} + 7 \Delta (\sin 2^{-n}) \quad (IX-65)$$

Results of these simulations would be valuable in further evaluating digital Stieltjes integration processes and associated algorithms.

CHAPTER X

**QUANTITATIVE EVALUATION OF COMPUTATION AND RATE HANDLING
CAPABILITY OF DDA AND QDDA MECHANIZATIONS**

10.0 GENERAL DISCUSSION OF QDDA COMPUTER MECHANIZATION ALTERNATIVES FOR DIFFERENT COMPUTATION TASKS - The major problem of quantitative evaluation of computation capacity, in relation to assumed incremental computer mechanization features, was attacked. Before proceeding to the quantitative evaluations, consider a number of heuristic aspects of design associated with mechanization. The results of the computer application and computation requirement study demonstrate that the various airborne and aerospace application computations fall into two fairly distinct classes with respect to their computation requirements (associated with their input processing and internal computation requirements). The implication of this result is that the incremental computer, assigned an overall computation task consisting of a given set of computation routines all of which fall into the same distinct group, should be evaluated for distinct mechanizations meeting the pertinent computation requirements to determine the most economical mechanization. For the case where the computation task includes computations of both distinct computation types, a deeper design problem is presented in mechanization optimization. This problem is encountered in the development of a computing system for a full aerospace mission requiring input processing capabilities. Thus, the analysis here is a quantitative step in the ultimate quantitative function-mechanization analysis of computing systems for full mission. The computation requirements which characterize the pertinent classification of the computation types stem from the occurrence or non-occurrence of variables of the real-time computation problem which vary per unit time in such great magnitude, in relation to required resolution, that special computer design features are required

to mechanize a computer which can execute the computation task. The occurrence of variables of this type are said to present the rate handling problem to DDA operation. The special computer design features which can obviate the rate handling problem are relatively high computation iteration rate (as obtained by processing efficiency and parallelity and/or high clock-rate) and/or multi-increment computation. Computation tasks for a total mission which do not, in the normal sense, present a rate handling problem, (computation quantity variation per unit time does not directly imply unacceptable resolution), are nevertheless typically characterized by another computer design problem of comparable proportion to the rate handling problem. The second problem consists of the twin problems of assuring ability to execute a program consisting of a relatively large set of different computations, and that of long term computation accuracy. Design features required to meet what may be called the computation capability problem are: optimized algorithm and multi-increment computation for precision; processing versatility and parallelity for efficiency; and high iteration rate for problem scope and accuracy. Clearly, from the standpoint of abstract processings (apart from mechanization costs), the design problems of rate handling and computation capability may be solved together in the design of a single computation ensemble. In the practical case where the level of mechanization complexity is a priori assigned, there exist mechanization alternatives which enable handling computation tasks involving the problem of rate handling capability in the one case, and in the other case the problem of computation capability (the latter in the sense of not including in the fullest degree the problem of rate handling capability). The basic reason, in the case of incremental computers, for the possibility of distinct mechanizations for best meeting the distinct computation problems, lies in the relation of mechanization complexity limitation and iteration rate for multi-increment computation in conjunction with the relation of iteration rate and computation capacity. A mechanization of the more modest complexity level which achieves multi-increment

computation does so only at reduced iteration rate (in consequence of increased serialized operation) which in turn implies a source of reduced computation capacity in considerable degree offsetting the associated ability to increase accuracy which is implied by the selected multi-increment feature. For the computation class presenting a computation capacity problem rather than a rate handling problem, the allocation of hardware totaling the a priori level of mechanization complexity may be chosen to achieve increased processing capacity while meeting readily attainable accuracy requirements of particular application types, rather than simply the pure alternative allocation to obtain multi-increment computation. On the other hand, computation tasks presenting the problem of rate handling are executed with highest performance by allocating the a priori level of mechanization complexity to multi-increment design in consequence of the relatively large increase in rate handling capability implied by multi-increment computation relative to that attainable by resorting to a high degree of processing parallelity. In consequence of the relations of mechanization complexity, processing structure and computation accuracy discussed above (and given quantitative description in the following section), a number of computer types are defined and analyzed and evaluated for specific tasks occurring in airborne and aerospace applications. In particular the multi-increment QDD³A involves all the design developments of the contract study and is capable of all computation tasks investigated in the applications study. In consequence of a degree of parallelized processing and the 3 bit transfer features, and input processing capability with 6 bit increment is obtained by an economical modal switching. Numerical evaluations for the latter are obtainable by the formulae developed. Examples, however, were confined to internal computation. While certain modest computation tasks may be most economically mechanized in alternative forms discussed above (actually based on the proposed computer) and analyzed in the following pages, the many and varied total mission requirements can only be met by a computer on the level of sophistication of the proposed parallelized multi-increment QDD³A with input processing capability.

10.1 FORMULAE FOR COMPUTER COMPUTATION CAPACITY AND RATE HANDLING CAPABILITY IN TERMS OF MECHANIZATION FEATURES - The programmable QDPU, which is a generalization of the DDA integrator, effects according to specific mechanization features a given number of B bit multi-transfers within the cycle of its action. The completion of which is followed by the next processing by the same hardware logic according to the generally modified program on different stored information with different inputs of what may be called the next QDPU of the block. Depending on the specific design, a type of average performance of the QDPU for an application or set of applications may be evaluated with respect to the average number of conventional DDA integrators which would be required to program the same application or set of applications; this average performance will be termed the average integrator equivalent N_{I_e} of the QDPU. The conventional integrator requires one word time to be processed whereas the QDPU may, according to chosen mechanization, require N_W word times. The average integrator equivalent per word time, $R = N_{I_e}/N_W$, is the QDPU machine processing rate per word time of the QDDA*. Actually, computer computation capacity is best measured by the number of different integrators equivalent which can be processed with required accuracy for a given application or set of applications. Hence, accuracy performance is closely related to computation capacity. If the longest tolerable time interval between processing the same QDPU (or DDA integrator is τ_Q (or τ_D) such that required accuracy is obtained, then the computation capacity C for QDDA and conventional DDA is

$$\text{QDDA: } C_Q = R \cdot \tau_Q / \tau_W = \frac{N_{I_e} \cdot \tau_Q}{N_W \cdot \tau_W} \quad (\text{X-1})$$

*Word time τ_W will be taken to be determined by the same state of the art hardware, the same for QDDA and conventional DDA.

DDA:

$$C_D = 1 \cdot \tau_Q / \tau_W = \frac{\tau_D}{\tau_W} \quad (X-2)$$

The relative computation capacity of QDDA (relative conventional DDA without programmable integration algorithm) is the ratio

$$C_{\text{relative}} = C_Q / C_D = \frac{N_{I_e} \cdot \tau_Q}{N_W \cdot \tau_D} \quad (X-3)$$

The importance of state of the art word rate in determining the adequacy of a computer, makes absolute computation capacity the important quantity. The QDPU information storage and processing configuration developed during this contract study has $N_{I_e} \approx 4.3$. Mechanization complexity determines the selected value of $N_W = 1, 2, 3, 4$. In the next section a table presents estimates of iteration interval required for specified accuracy for a number of applications and hypothesized computer types based on the theory of computer error growth derived in this study. The above mentioned data were used in computing the computation capacity of various computer types, and applications for the stated word rate (obtained by state of the art hardware). These results are shown as follows:

Computation Capacity (Integrators Equivalent) for Application Routine
in the Case of 1.4×10^6 Word Rate (for other word rates multiply
tabulated values directly by word rate ratio)

Frequency of Variable f	f=.4 cycles per hr	f=.05 cps t=15 min	f=.16 cycles per hr	f=.015 cps t=1000 sec	f=.1 cps t=2 min
Computation Price of Routine Accuracy Requirement, t_0	t=1.5 hr	$\epsilon=1 \times 10^{-3}$	t=10 hr	(damping time)	$\epsilon=.6 \times 10^{-3}$
	$\epsilon=10^{-8}$	$\epsilon=10^{-4}$	$\epsilon=10^{-4}$	$\epsilon=10^{-2}$	
Computation Capacity tabulated lower right for:	Orbital calc. (excluding mid-course guid., propulsion, stages, reentry)	Midcourse guid., propuls., stages, entry	Conventional navigation (excluding gation: high frequency Doppler loops)	Conventional Airborne Inertial Navigation	Toss Bombing
Programmable Algorithm	13. integ eq .6 integ eq	.4 integ eq	280. integ eq	7. integ eq	.4 integ eq
1 bit increment QDDA	180. integ eq	210. integ eq	1100. integ eq	2400. integ eq	96. integ eq
	$(N_p = 1)$				

**Computation Capacity (Integrators Equivalent) for Application Routine
in the Case of 1.4×10^4 Word Rate (for other word rates multiply
tabulated values directly by word rate ratio) (cont)**

Frequency of Variable f f=4 cycles	f=.05 cps t= 1.5 min e= 1 x 10 ⁻³	f=.16 cycles per hr t= 10 hr e= 10 ⁻⁴	f=.015 cps t= 1000 sec (damping time) e= 6 x 10 ⁻³ f=10 ⁻³
Computation Price of per hr			
Routine Accuracy t= 1.5 hr			
Requirement, t _e e= 10 ⁻³			
2 bit increment (N _w = 1) 300.	420.	180.	
3 bit increment (N _w = 1) 660.	600.	360.	
4 bit increment (N _w = 2) 660.	450.	300.	
4 bit increment (N _w = 1) 1320.	900.	500.	

Note: A full mission computer has a number of computation tasks which must be evaluated taking into account program time sharing. This may be done employing effective word rates as the fraction of hardware word rate equal to the fraction of time in subroutine.

Rate handling capability f_{\max} may be taken to be measured by the maximum frequency f of variable which can be scaled with a given resolution R . Since rate of change of the variable in one iteration time determines the maximum resolution, the iteration rate of the computer is pertinent to rate handling capability, as is the multi-increment level. Thus, the rate handling capability may be expressed in terms of the maximum frequency of sinusoid calculable with given resolution which is

$$f_{\max} = (2^M - 1) / 2 \pi \tau R \quad (X-4)$$

where $\omega = 2 \pi f$. The iteration interval is actually determined by required computation capacity for given mission accuracy requirement. However, consider a more abstract formulation* of relative f_{\max} of various computers which simply assumes equal integrator equivalent count ignoring long term accuracy performance (this abstraction ignores, for example, the superior algorithm accuracy of the QDDA relative conventional DDA). In the latter case

$$\frac{f_{\max}}{\text{Relative Conventional DDA}} = (2^M - 1) \frac{\tau_D}{\tau_Q} \quad (X-5)$$

where τ_D/τ_Q is taken now as relative iteration interval for the same integrator equivalent count assuming identical word rate of the hardware. In terms of the previously defined variables of mechanization alternatives,

$$\frac{\tau_D}{\tau_Q} = \frac{N_I}{N_w} \quad (X-6)$$

*In contrast to computation capability as analyzed above, regarded as the ultimate quantitative evaluation of computer performance, the rate handling capability is regarded a heuristic quantity in this analysis.

hence, the QDDA has relative rate handling capability

$$f_{\max}^{\text{Relative Conventional DDA}} = (2^M - 1) \frac{N_I}{N_w} \quad (X-7)$$

which using $N_I \approx 4.3$ implies:

$$\text{Single increment QDD}^3\text{A with } N_w = 1 \quad f_{\max}^{\text{Relative DDA}} = 4.3$$

$$\text{3 bit increment QDD}^3\text{A with } N_w = 1 \quad f_{\max}^{\text{Relative DDA}} = 30. \text{ (QDD}^3\text{A Single Precision Mode)}$$

$$\text{6 bit increment QDD}^3\text{A with } N_w = 2 \quad f_{\max}^{\text{Relative DDA}} = 135. \text{ (QDD}^3\text{A Double Precision Mode)}$$

assuming identical integrator equivalent count of both QDDA computers.

10.2 PRECISION LEVELS RESULTING FROM ALGORITHM OF QDDA AND CONVENTIONAL DDA - The precision level at a given iteration rate is determined by integration algorithm and roundoff mechanization. The errors magnitudes produced are a function of iteration rate and period of operation. The conventional DDA does not have programmable algorithm and suffers thereby, in certain computations, integration algorithm errors of first order; that is, the deviations which would result if trapezoidal, new y, old y iteration algorithms were arbitrarily permuted throughout a computation. DDA computations using servo modes introduce much larger errors by a factor f_s where f_s is believed to be about 4. The QDDA having programmable algorithm does not have integration algorithm errors of first order, but instead, errors of the second order from fractional approximation error δ of the mechanization of

second order integration algorithm terms (for $\beta = 0$ the algorithm is good to 2nd order, for $\beta = 1/2$ the algorithm is good only to 1st order). The quantitative error levels resulting from integration algorithm mechanization which will be typically encountered are

$$\text{Conventional DDA: } e = 1/2 \omega^2 \tau t f_s \quad (f_s = 4 \text{ in servo mode,} \\ f_s = 1 \text{ otherwise}) \quad (\text{X-8})$$

$$\text{QDDA: } e = 8\omega^2 \tau^2 t \quad (\text{X-9})$$

where

$\omega = 2\pi f$, f = frequency of computation variable

τ = iteration interval between computation updating

t = time interval of operation

f_s = factor of error increase in servo modes of the DDA

Another kind of computation error is roundoff error. Conventional DDA computers have both binary and ternary mechanizations. The design trend is toward ternary which has one half the roundoff error (associated with R register content relative that of binary mechanization), and has the other accuracy advantage of being relatively free of the "phase" effects which occur in binary computation (resulting from representation of zero in binary as the alternating stream). Evaluation of QDD³A accuracy will be relative to the conventional ternary DDA and hence tends to be conservative. A principle advantage of the QDD³A, with respect to roundoff effects, stems from the much reduced number of roundoff error sources, especially those which contribute the largest elements of error, and from the overflow inhibitor mechanism for the single increment QDD³A and otherwise multi-increment computation. In absolute count, the number of R registers in the QDD³A is less than one-half the count of the DDA for the same application. In the QDD³A for operations

not involving division the roundoff error is reduced by a factor of two so far as R-register count. In operations involving division algorithm, the effective reduction of roundoff error is much larger since a DDA program effecting division involves several integrators acting either as a servo loop subject to relatively large errors compared to ordinary roundoff error, or acting as Stieltjes integrators with integrand and independent variables subject to spasmodic error effects comparable in magnitude to the phase error effects troublesome in a binary DDA. In division modes (without servo), simulations of conventional DDA show that errors are generated which have a behavior markedly similar in magnitude to those produced by integration algorithm errors of first order; hence, such is assumed in the quantitative estimates below. The absolute magnitude of roundoff errors depends on scaling of the computations in all cases. In general, the rate limit character of general DDA computation forces the minimum granularity to be at least as large as that consistent with maximum rate of change of the variable; hence, $\frac{\pi \tau}{2\sqrt{3}} \cdot 2^{-M}$ is the minimum rms magnitude of roundoff associated with resolution for M bit increment (assuming a flat distribution of error). Evidently, the high frequency variables of the computation task are the most subject to roundoff error, as is also true for integration algorithm error. If for the moment we regard roundoff error as a kind of noise effect in computations not involving division, the error growth factor is deduced to be $\sqrt{n} = \sqrt{t/\tau}$, whereas in computations involving division in the conventional DDA the error growth factor is $n = t/\tau$.

The attempt to explain roundoff error magnitude for the growth (not the resolution) term in the DDA by regarding instantaneous roundoff error as a random independently distributed variable (such that the integral over n iterations of instantaneous roundoff error has the rms value of $\frac{\pi \tau}{2\sqrt{3}} \sqrt{n}$) is found to far over estimate the error which actually occurs. The reason is that the roundoff errors forcing function series in time does not have genuine independent component errors, but rather has the property that their sum over a period is of

the same order of magnitude as the typical component, a consequence of the residue retention property of the R register. In a system of integrators, there can occur growth of roundoff error in consequence of transient states of roundoff error during which the system sensitivity to error forcing function is first in one sense and then the other in phase with the reversal (having resulted from the correlative compensation effect) of the roundoff series sum which leads to net error buildups, rather than cancellation over the subsequent period. Thus, in sinusoid calculation the system sensitivity oscillates and when the roundoff error forcing function has transient oscillatory components of the same frequency as the sinusoid, a net error growth ensues despite the long term cancellation of the sum of roundoff error forcing functions. Over much longer periods, the unsteady length of the correlation period would produce random long term phase cancellations implying a \sqrt{t} growth (where t is time) in rms error. The largest resonance effect occurs near matched frequencies, specifically when the y register has near null value, since the time interval for correlative reversal of roundoff is stretched to approach that of the system response (computation weighting function). As an exception, there may be, for short registers, other periods of strong resonance by chance diophantine relation of y register content such that R register tends to be biased in one sense for considerable periods. Scaling for maximum resolution presents the instantaneous roundoff error of rms magnitude $\frac{\omega \tau}{2\sqrt{3}}$ which is of the same order of magnitude as the instantaneous increment of 1st order algorithm error. Over long periods, growth of roundoff error may be associated with that of a random noise model in which error sensitivity occurs only during the near null states of the y registers. The resulting formulation is approximately

$$\epsilon = \left(\frac{\omega \tau}{2\sqrt{3}} \right) \sqrt{1 + \frac{8}{\pi^2/2\omega\tau}} \quad (X-10)$$

where the first term is simply the resolution for optimum scaling. The round-off error implied by this formula is consistent in magnitude with DDA sinusoid simulation results during this program; its formulation is consistent with the observation that sporadic roundoff effects occurred during null passes of the y registers. The overflow inhibitor, developed during phase 1 of the contract study, had the property of reducing the sporadic phenomenon by a factor of at least 3, hence has been chosen as a design feature of the single increment QDDA. The reduced R register count reduces the error another factor of two.

Summarizing, the roundoff error performance of QDDA and conventional ternary DDA implied by above considerations the approximate relations are:

Roundoff Error of Conventional DDA

$$\left\{ \begin{array}{l} \text{No division calculation: } e = \frac{\omega T}{2\sqrt{3}} \sqrt{1 + \frac{\omega t}{\pi/2}} \\ \text{Division and servo calculation: } e = \frac{\omega T}{2\sqrt{3}} [1 + \sqrt{3}\omega t] f_s \end{array} \right. \quad (X-11)$$

$$(X-12)$$

Roundoff Errors of QDDA

$$\left\{ \begin{array}{l} \text{Single increment: } e = \frac{\omega T}{2\sqrt{3}} \sqrt{1 + \frac{1}{6} \frac{\omega t}{\pi/2}} \\ \text{overflow inhibitor: } \\ \text{M bit increment (M ≥ 3) } e = \frac{\omega T}{2\sqrt{3}} 2^{-M} \sqrt{1 + \frac{1}{6} \frac{\omega t}{\pi/2}} \end{array} \right. \quad (X-13)$$

$$(X-14)$$

The total error, including integration algorithm and roundoff error, is (where component errors actually enter as sum of squares, but to save space in later analysis, are entered linearly without appreciable error for present purposes):

Conventional DDA: $e_{\text{total}} = \frac{\omega T}{2\sqrt{3}} \left[\sqrt{1 + \frac{\omega t}{\pi/2}} + \sqrt{3}\omega t \right]$ Error in 1st order algorithm (X-15)

No division calculation: $e_{\text{total}} = \frac{\omega T}{2\sqrt{3}} \left[\sqrt{1 + \frac{1}{6} \frac{\omega t}{\pi/2}} + \sqrt{3}\omega^2 T t \right]$ Programmable 1st order algorithm (X-16)

Division & servo
calculation:

$$e_{\text{total}} = \frac{\omega T}{2\sqrt{3}} [1 + f_s \sqrt{3} \omega t] \quad (\text{X-17})$$

QDDA:

Single increment &
overflow inhibitor:

$$e_{\text{total}} = \frac{\omega T}{2\sqrt{3}} \left[\sqrt{1 + \frac{1}{6} \frac{\omega t}{\pi/2}} + \sqrt{3} \omega^2 \tau_t \right] \quad (\text{X-18})$$

Multi-increment
(M bit):

$$e_{\text{total}} = \frac{\omega T}{2\sqrt{3}} \left[2^{-M+1} \sqrt{1 + \frac{1}{6} \frac{\omega t}{\pi/2}} + \frac{\sqrt{3}}{6} \omega^2 \tau_t \right] \quad (\text{X-19})$$

For typical programs of conventional DDA, variables which go through substantial variation during operating time have error magnitude of the order

$$e_{\text{total, DDA}} \approx \frac{\omega^2 \tau_t}{2} f_s \quad (\text{X-20})$$

The error in QDDA from integration algorithm is less than this by the factor $\frac{\omega T}{f_s}$ in the single increment machine, and $\frac{\omega T}{6f_s}$ in the multi-increment machine.

At very low frequencies the QDDA error is mainly roundoff error, typically of the order $\omega T \cdot 2^{-M-1}$

The computation capacity is computed in terms of the minimum iteration interval required to meet specified accuracy. For the QDDA, the pertinent minimum iteration interval τ_Q is given approximately by the quadratic

$$0 = -c + a\tau_Q + B\tau_Q^2 \quad (\text{X-21})$$

where

$$a = \frac{\omega}{\sqrt{3}} \cdot 2^{-M} \sqrt{1 + \frac{1}{6} \cdot \frac{\omega t}{\pi/2}}$$

$$\theta = \frac{\omega^2 t}{2(1 + 5u(M - 1^+))}$$

the u function be 0 for single increment, 1 for multi-increment. The solution to the quadratic,

$$\tau_Q = \frac{a}{2\theta} \left[\left[\sqrt{1 + \frac{e}{\epsilon_0}} - 1 \right] \right] \quad (X-22)$$

where $\epsilon_0 = a^2/4\theta$, is simplified in two ranges of specified accuracy separated by ϵ_0 where approximations hold based on $(\sqrt{1+x} - 1) \approx \frac{x}{2}$ for $e \ll \epsilon_0$ or $(\sqrt{1+x} - 1) \approx \sqrt{x}$ for $e > > \epsilon_0$ which relate physically to the predominance of roundoff error or integration algorithm error. Thus

$$\tau_Q \approx \frac{e}{a} \text{ for } e \ll \epsilon_0 \quad \text{Roundoff Predominant} \quad (X-23)$$

$$\tau_Q \approx \sqrt{\frac{e}{\theta}} \text{ for } e > > \epsilon_0 \quad \text{Integration Algorithm Predominant} \quad (X-24)$$

where

$$\epsilon_0 = \frac{[1 + 5u(M - 1^+)]}{6 \cdot 2M} \quad \left(\frac{1}{3\pi} + \frac{1}{\omega t} \right)$$

These relations for minimum iteration rate of the QDDA may be written in the form

$$\omega t \tau_Q \approx \frac{e \cdot 2^M \sqrt{3}}{\sqrt{1 + \frac{1}{6} \cdot \frac{5R}{\pi/2}}} \quad \begin{aligned} &\text{for } e \ll \epsilon_0 \\ &\text{(Roundoff Predominant)} \end{aligned} \quad (X-25)$$

$$w^T_Q \approx \sqrt{2e (1 + 5x(M - 1^+))} \quad \text{for } e >> \epsilon_0 \quad (\text{X-26})$$

(Integration Algorithm
Error Predominant)

where

$$\epsilon_0 = \frac{[1 + 5x(M - 1^+)]}{6.2^{2M}} \left(\frac{1}{3\pi} + \frac{1}{wt} \right)$$

The relation for minimum iteration rate of conventional serial DDA is

$$w^T_D \approx \frac{2e}{wt f_s} \quad (\text{X-27})$$

A table of minimum required iteration rate* for conventional DDA, 1, 2, 3, 4 bit increment QDDA, has been calculated for several markedly different applications.

*The values express requirements for accuracy independent of attainability at state of the art bit rate.

ESTIMATED ITERATION INTERVAL REQUIRED FOR APPLICATION ROUTINES
(MECHANIZATION CAPABILITY NOT CONSIDERED)

Frequency of Variable, f ; $f=4$ cycles	$f=0.05$ cps	$f=.16$ cycles per hr	$f=.015$ cps	$f=.1$ cps
Computation Period of Routine, t ; Accuracy	$t=1.5$ hrs	$t=1.5$ min	$t=1000$ sec	$t=2$ min
Requirement, ϵ	$\epsilon=10^{-5}$	$\epsilon=1 \times 10^{-3}$	$\epsilon=10^{-2}$	$\epsilon=.6 \times 10^{-3}$
Minimum iteration interval (tabulated lower right) for:	Orbital calc	Midcourse	Conventional	Toss
	(excluding mid-course guidance, propulsion stages, reentry entry)	Airborne Inertial Navig.	Airborne Inertial Navig.	Airborne Inertial Navig.
	(Excluding high Doppler freq. loops)	(Excluding high Doppler freq. loops)	(Excluding high Doppler freq. loops)	(Excluding high Doppler freq. loops)
Conventional DDA (1st order integration algorithm error)	$\tau=.4 \times 10^{-4}$ sec*	$\tau=.3 \times 10^{-4}$ sec*	$\tau=.02$ sec	$\tau=.5 \times 10^{-3}$ sec*
			Servo	.2x10 ⁻² sec*
				$f_g=4$ Servo
				$f_g=1$ Div.
Programmable Algorithm	$\tau=.9 \times 10^{-3}$ **	$\tau=.8 \times 10^{-3}$ **	$\tau=.08$ Div.	$\tau=.3 \times 10^{-4}$ sec **
DDA (correct 1st order algorithm; no division mode like that of QDDA)				
				2×10^{-3} , $f_g=1$ Div.

*Primary error source is integration algorithm.
**Primary error source is roundoff error.

**ESTIMATED ITERATION INTERVAL REQUIRED FOR APPLICATION ROUTINES
(MECHANIZATION CAPABILITY NOT CONSIDERED)(cont)**

Frequency of Variable, f; f=4 cycles per hr	f=.05 cps t=15 min	f=.16 cycles per hr	f=.015 cps t=1000 sec
Computation Period of Routines, t; - Accuracy Requirement, e	t=1.5 hrs e = 10^{-6}	t=10 hr e = 10^{-4}	(damping time) e = 10^{-3} e = 6×10^{-3}
1 Increment QDDA	$\tau = .3 \times 10^{-2}$ **	$\tau = 3.5 \times 10^{-3}$ **	$\tau = .8$ sec **
2	$\tau = .5 \times 10^{-3}$ **	$\tau = .7 \times 10^{-3}$ **	$\tau = .08$ **
3	$\tau = 1.1 \times 10^{-3}$ **	$\tau = 1 \times 10^{-3}$ **	$\tau = .16$ **
4	$\tau = 2.2 \times 10^{-3}$ **	$\tau = 1.5 \times 10^{-3}$ **	$\tau = .32$ **
			$\tau = 1.0 \times 10^{-2}$ **

*Primary error source is integration algorithm.
**Primary error source is roundoff error.

CHAPTER XI

MULTI-INCREMENT QDD²A PROGRAMS FOR IMPORTANT APPLICATIONS AND PROGRAMMING EFFICIENCY EVALUATION

**11.0 THE GENERAL LEVEL OF COMPUTER SOPHISTICATION
REQUIRED FOR FULL AEROSPACE MISSION** - The general level of computation capability and associated mechanization complexity required of a computer for a full aerospace mission is determined by the program task and state-of-the-art bit rate. The effort here is to evolve an incremental computer in a GP-DDA computer system which takes over the bulk of the computation tasks of the aerospace mission and leaves the remaining functions to a low cost GP with slow multiplier.* The proposed approach is an allocation of the computation tasks for a computer system of maximum efficiency, based on the individual potentialities of the two computer types. Evaluation during this study of the explicit computation programs for a full (or almost full) aerospace mission, evolved in other contract efforts at Litton Systems, Inc., demonstrates that an incremental computer system, of the remarkable new type evolved in this contract study, must be capable of carrying out a program (the extent of which is expressed in conventional DDA integrators) for 400 to 500 DDA integrators if no branching is mechanized, and 250 DDA integrators with branching. Significant portions of such a system require new levels of computation accuracy relative to existing DDA computers. Hence, efforts toward evolving serial-parallel computer structure of the QDD²A were concentrated on mechanizations having a complexity intermediate between conventional DDA and an airborne GP computer. By virtue of basic developments in multi-increment computation and modal action of computer processing structure, significant

*Early efforts in the aerospace computer field have apparently confined consideration to computing systems in which a conventional DDA is not capable of executing a substantial part of the program task and requires a large GP with clock rate pressing the state-of-the-art.

reductions in computer complexity for a given level of computation capability were recognized as attainable. The system design task of exploiting these potentialities for an incremental computer capable of assuming the major tasks of a full aerospace mission involves a combination of application programming and mechanization studies. The former are given major emphasis in this chapter. The quantitative analysis of computation capability as a function of design features of Chapter X and the preceding investigations of general input processing mechanizations requirements for state-of-the-art bit rates imply a computer with paired many-bit transfer and double paired several-bit transfer mechanization alternatively programmed in a time shared arithmetic structure.

Programmable modal action of the QDD³A was developed in closely parallel investigations of, on the one hand, application computation requirements, and on the other, development of digital processing and arithmetic modal design features. In this chapter, explicit programs for important applications are developed and/or evaluated in order to evaluate the high computation capability of the product for its intended application. The discussion stresses that application requirements are specifically met by basic operations. A preceding chapter presented a description of the digital processing and arithmetic modal design techniques which make the basic operations possible in a computer of efficient hardware mechanization. Thus, here we assume the programmability of modes of single and double precision, high and intermediate iteration rate, (which make the general computations of input processing and internal computation types possible with the same array of flip-flops time shared and continuously fully worked) to achieve maximum computation capability. In internal computation, where precision is traded to achieve effectively high speed computation by computation operation sophistication, we assume the programmability of transfer and algorithm action defined by the QDD³A program code in Chapter XII.

The major application for which QDD²A programmability is evaluated is the full aerospace mission, including the many phases of guidance, allowing for undetermined auxiliary functions of military context conceived likely within the coming decade in order to provide a basis for assurance that the latter expected additional functions may be within the computation capability of the computer (clearly beyond the capability of any existing airborne incremental computer) when added to the already large guidance and control program. Certain other computations are evaluated which are known to challenge, or clearly exceed in certain respects, the capabilities of conventional computers. Thus, the important computations for doppler damping in airborne navigation are presented because satisfactory damping accuracy is recently known to be unsatisfactory on the basis of flight test (and also analysis of Chapter X). The reasons for this particular failure stems from a combination of sinusoid generation inaccuracy for rapidly changing craft angles and accuracy deficiency of division in the conventional DDA. Clearly, certain military applications such as fire control may present to an aerospace application of the future similar computational demands as those of doppler damping and toss bombing. Input processings for the case of strap-down computations and midcourse guidance are analyzed.

The relations of input processing and internal computation as computation types (general concepts for which were developed during phase I) shall be delineated in terms of later developed modal design principles and mechanization features. The essence of input processing requirements is the requirement for many-bit increment computation and high iteration rate, in respective degrees to attain a required resultant capacity. The developed mechanization features for the many-bit increment computation is programmable at high or intermediate iteration rate; the iteration rate being assignable provided program total is not excessive. The mechanization this obtains is: (1) many-bit increment computation (double

precision) at high rate to be selected for the most demanding routines; (2) many-bit increment at intermediate rate for somewhat less demanding routines, which it should be added, tend to be too long to be performed at a very high rate; and (3) several-bit increment computation at high rate for computations requiring rapid decision such as thrust cutoff after mid-course guidance. The several-bit increment computation at very high rate is entirely adequate for the best pulse stream transducers. Because of increased parallel computation of the QDPU, over that in double precision mode, high iteration rate is made compatible with attaining a satisfactory intermediate iteration rate due to reduced time sharing requirements; (4) several-bit increment computation at intermediate iteration rate for typical internal computations. The input processing computations programmed and analyzed in detail in this applications study may be listed as follows:

Input Processing Programs

High Rate Single Precision:

- (1) Thrust Cutoff: For full aerospace mission
- (2) Strap-down: For full aerospace mission

Intermediate Rate Double Precision:

- (1) Sine and cosine computations on Air Data Inputs exemplifies re-entry, fire control, digital autopilot computation subroutines
- (2) Doppler Damping of inertial systems: Enables long term airborne navigation accuracy.

The problem of handling 90% to 100% of the computer system (GP-DDA) task in the incremental computer is the two-fold problem of: (1) computation capability for well behaved functions, and (2) evolving decision mode mechanization and programming techniques for handling discontinuities and singularities (the latter being usually a result of coordinate system).

The first part of this problem is resolved by the computer design principles developed during phase II and specifically by the QDD³A. The second problem, allocation of effort to which was limited within the scope of this study, deserves intense study. The preliminary study made of this problem indicates that it is not insurmountable and can be eventually achieved in relatively efficient mechanization.

11.1 QDD³A PROGRAM ANALYSES AND COMPUTER SYSTEM DESIGN BASED ON COMPUTATION REQUIREMENTS FOR AEROSPACE GUIDANCE AND CONTROL APPLICATIONS

A. INTRODUCTION - In order to assure that computer design offered in satisfaction of this study contract implies a mechanization capable of serving the total requirements of an orbital mission, the set of computations presented in "Formulation of Guidance and Control Equations, Their Mechanization and Instrumentation" (October 1961) by W. J. Jacobi and C. S. Bridge have been selected as the minimal computation requirements thereof. The orbital mission is postulated to have 8 modes:

1. Boost Phase
2. Coast to Apogee (Transfer Orbit)
3. Injection into Orbit
4. Orbit
5. Retrofire
6. Transfer Orbit
7. Re-entry (Development of Aerodynamic Forces)
8. Manuevering and Landing

The computation requirements delineated in the pertinent paper consider a guidance and control system comprised of the following:

1. Inertial Platform
2. Central Digital Computer
3. Environment Sensors and Buffers for Communication with the Outside World

A central digital computer is presented minimal requirements for the full mission as a result of assumption (1) above; thus, if it is assumed that strap-down navigation (without inertial platform) is implemented, or if functions other than guidance and control are to be implemented, then the computer system must have a computation capacity in excess of that implied in the pertinent paper. Major system output functions for guidance and control are:

1. Velocities in earth fixed and space fixed coordinates
2. Altitudes above earth's mean surface
3. Angles relative to local vertical
4. Angular position in earth fixed coordinates
5. Automatic flight control information
6. Energy management parameters

The multi-increment QDD²A is designed to have a QDPU (quotient differential processing unit as a generalization of the conventional DDA integrator) capable of an unparalleled level of speed and precision in performing elementary incremental computations such as multiplication, vector resolution, division, $\sqrt{x^2 + y^2}$, scaling, integration, sinusoid generation, and other DDA computations implied in the above functions (1) through (5) in an optimized mechanization derived by analyzing the relations of elementary computation requirements and associated hardware requirements. While the QDD²A design evolved is programmable for any set of incremental computations, the specific program for (1)

through (6) is largely blocked out for evaluation of the QDD²A computation capacity, in relation to existing incremental computers, and for potential application of QDD²A mechanization.

B. Missile Velocity Computation with Scale Factor, Linear Drift and Bias Error Correction of Inputs and Outputs of the Digital Computer - In order to achieve the accuracy required, the computer may be used to compensate for various scale factor, linear drift, and bias errors of input sensors and computer commanded transducers. An internal computation with input processing supplied inputs may also be required to effect scale factor changes. An inertial system with a physical inertial platform employing accelerometers subject to known individual errors is torqued to desired orientation by torquers subject to known individual errors. Execution of the desired error corrections may be shown possible carrying out the following calculations:

$$\Delta V_x = \Delta V_x' (SF_x) + b_x \Delta t \quad (\text{XI-1})$$

$$\Delta V_y = \Delta V_y' (SF_y) + b_y \Delta t$$

$$\Delta V_z = \Delta V_z' (SF_z) + b_z \Delta t$$

$$w_x' \Delta t = w_x \Delta t (SF_x') + b_x' \Delta t$$

$$w_y' \Delta t = w_y \Delta t (SF_y') + b_y' \Delta t$$

$$w_z' \Delta t = w_z \Delta t (SF_z') + b_z' \Delta t$$

where $\Delta V_x^i, y, z$, ω_x^i, y, z , Δt are input and output quantities respectively. The conventional DDA requires 6 integrators to effect the scale factor corrections and 6 integrators to effect the drift corrections of accelerometer inputs and angular rate outputs. In the QDD²A these corrections are blended with velocity computations to effect a marked increase in computation capacity. Before evolving the QDD²A program, consider the nature of the velocity computation best suited to incremental computation.

The velocity computation apart from scale factor, linear drift, and bias correction of inputs and outputs is derived from the general expression for acceleration in a rotating coordinate system,

$$\ddot{\mathbf{A}} = \ddot{\mathbf{R}} + \dot{\boldsymbol{\omega}} \mathbf{X} \ddot{\mathbf{R}} + 2\dot{\boldsymbol{\omega}} \mathbf{X} \dot{\mathbf{R}} + \dot{\boldsymbol{\omega}} \mathbf{X} (\dot{\boldsymbol{\omega}} \mathbf{X} \mathbf{R}) - \ddot{\mathbf{g}} \quad (\text{XI-2})$$

where $\ddot{\mathbf{g}}$ is the gravity vector. Substituting $\ddot{\mathbf{R}} = \ddot{\mathbf{k}} \mathbf{R}$ in the above equation and breaking the resulting vector equation into its components in the x, y, z directions yields:

$$A_x = \dot{\omega}_y R + 2\dot{\omega}_y \dot{R} + \dot{\omega}_x \omega_z R + g_x \quad (\text{XI-3})$$

$$A_y = -\dot{\omega}_x R - 2\dot{\omega}_x \dot{R} + \omega_y \omega_z R + g_y \quad (\text{XI-4})$$

$$A_z = \ddot{R} - (\omega_x^2 + \omega_y^2) R + g_z \quad (\text{XI-5})$$

These equations may be re-written in the following form suitable for computation by the QDDA*,

$$\Delta V_{x_n} = \Delta x_n - \frac{\text{Input}}{(n-1)^T} \int_{n\tau}^{(n-1)\tau} w_y V_z dt + \int_{(n-1)\tau}^{n\tau} w_z V_y dt - \int_{(n-1)\tau}^{n\tau} g_x dt \quad (\text{XI-6})$$

$$\Delta V_{y_n} = \Delta V_{y_n} + \frac{\text{Input}}{(n-1)^T} \int_{n\tau}^{(n-1)\tau} w_x V_z dt - \int_{(n-1)\tau}^{n\tau} w_z V_x dt - \int_{(n-1)\tau}^{n\tau} g_y dt \quad (\text{XI-7})$$

$$\Delta V_{z_n} = \Delta V_{z_n} - \frac{\text{Input}}{(n-1)^T} \int_{n\tau}^{(n-1)\tau} w_x V_y dt + \int_{(n-1)\tau}^{n\tau} w_y V_x dt + \int_{(n-1)\tau}^{n\tau} g_z dt \quad (\text{XI-8})$$

Interpreting $\Delta V_{x, y, z}$ as the scale factor, linear drift and bias corrected input quantities generated from actual inputs, the above equations and the correction equations presented previously

$$\Delta V_{x_n} = \Delta \lambda_x + \Delta \mu_x + b_x \Delta t - \int_{(n-1)\tau}^{n\tau} g_x dt \quad (\text{XI-9})$$

$$\Delta V_{y_n} = \Delta \lambda_y + \Delta \mu_y + b_y \Delta t - \int_{(n-1)\tau}^{n\tau} g_y dt \quad (\text{XI-10})$$

$$\Delta V_{z_n} = \Delta \lambda_z + \Delta \mu_z + b_z \Delta t - \int_{(n-1)\tau}^{n\tau} g_z dt \quad (\text{XI-11})$$

*But which will be further elaborated for scale factor, linear drift and bias correction of inputs and outputs.

where

$$\Delta \lambda_{x_n} = (SF_x) \Delta V'_x - \Delta \int v_z d \int w_y dt$$

$$\Delta \lambda_{y_n} = (SF_y) \Delta V'_y - \Delta \int v_x d \int w_z dt$$

$$\Delta \lambda_{z_n} = (SF_z) \Delta V'_z - \Delta \int v_y d \int w_x dt$$

$$\Delta u_x = \Delta \int v_y d \int w_z dt$$

$$\Delta u_y = \Delta \int v_z d \int w_x dt$$

$$\Delta u_z = \Delta \int v_x d \int w_y dt$$

The scale and bias corrected angular rates will also be computed in parallel QDPU action,

$$\Delta \int w'_x dt = \Delta \int (SF'_x) w_x dt + \Delta \int b'_x dt \quad (XI-12)$$

$$\Delta \int w'_y dt = \Delta \int (SF'_y) w_y dt + \Delta \int b'_y dt \quad (XI-13)$$

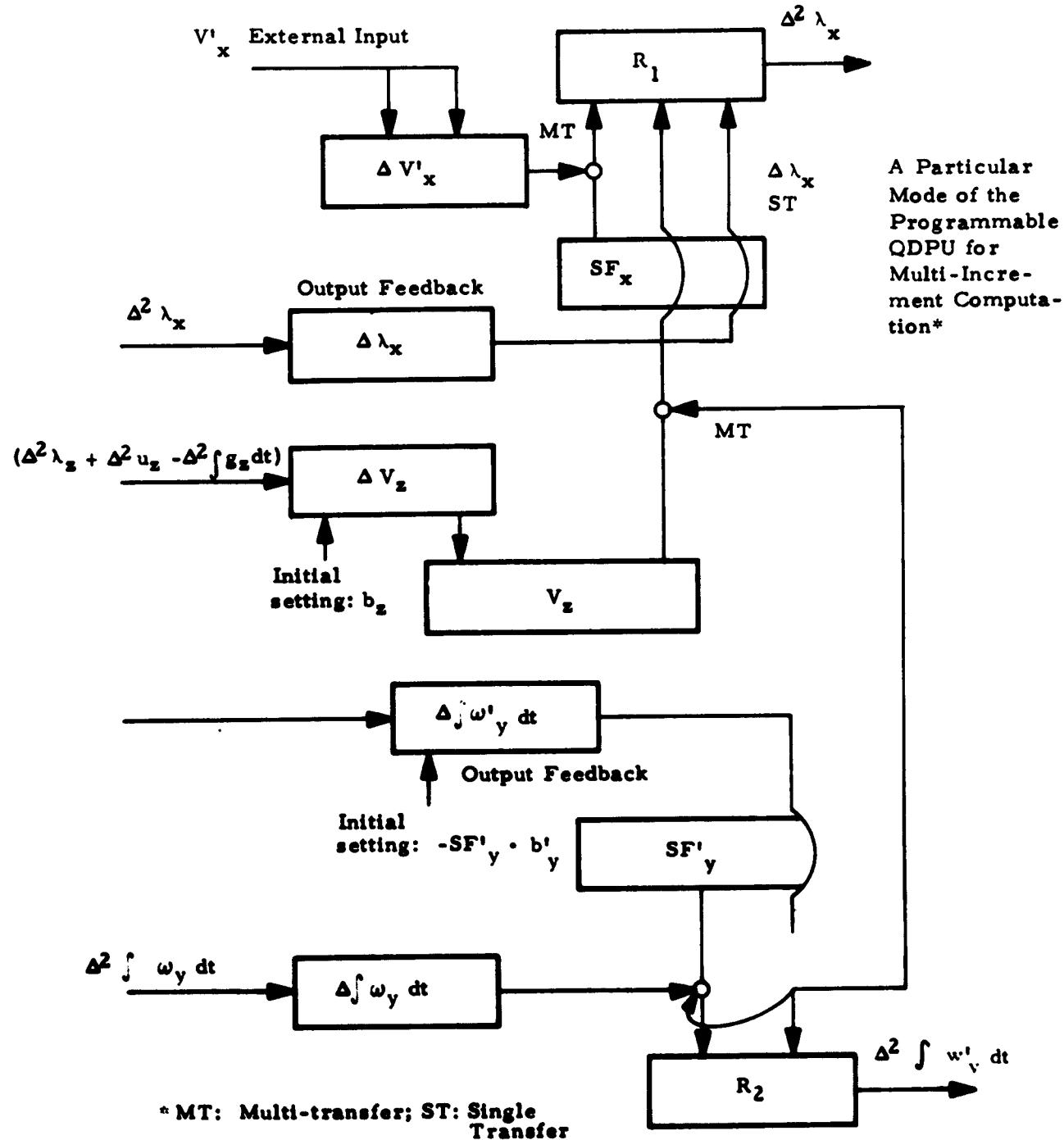
$$\Delta \int w'_z dt = \Delta \int (SF'_z) w_z dt + \Delta \int b'_z dt \quad (XI-14)$$

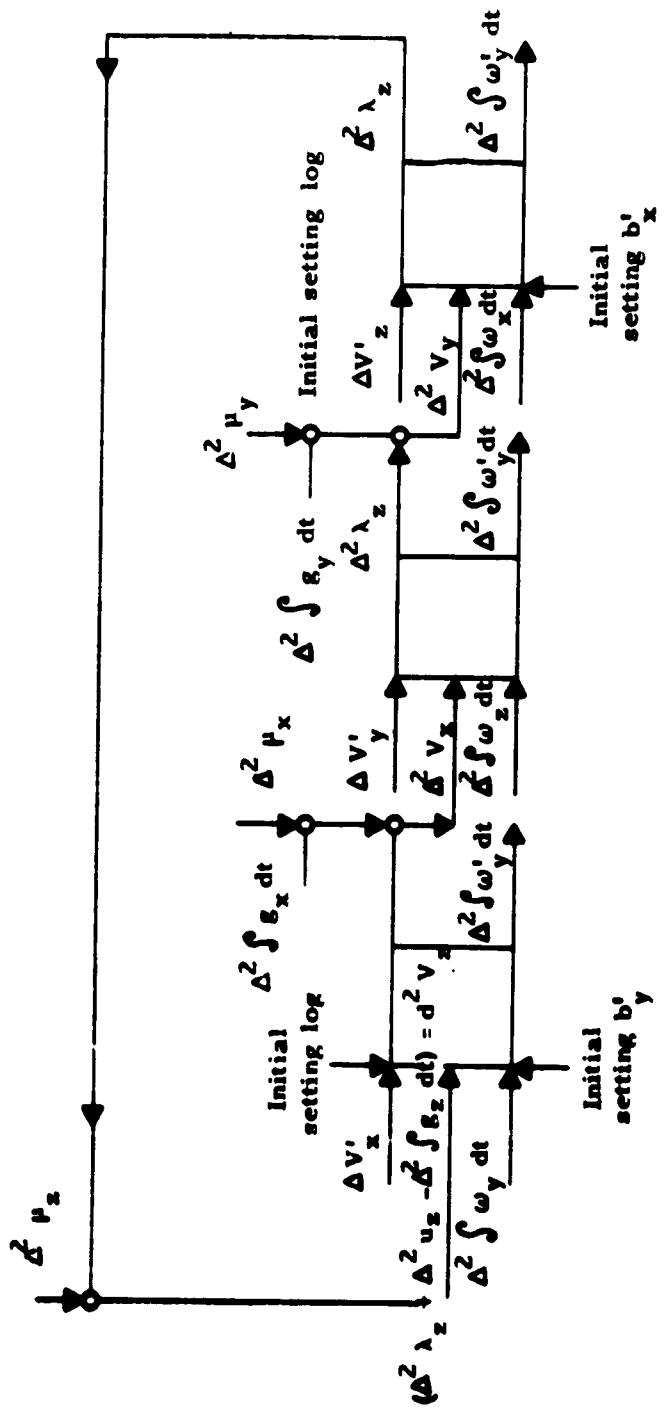
The quantities $\Delta V'_{x,y,z}$ are computer inputs. The quantities

$\int_{(n-1)^T}^{nT} g_{x,y,z} dt$ are generated in another routine which is developed.

The QDPU processing configuration for generation of $\Delta \lambda_{x_n}$, $\Delta \int \omega_y^t dt$ is typical for these types of terms and is shown in Figure 11-1.

Three QDPU generate $\Delta^2 V_x$, $\Delta^2 V_y$, $\Delta^2 V_z$ and $\Delta^2 \int \omega_x^t dt$, $\Delta^2 \int \omega_y^t dt$, $\Delta^2 \int \omega_z^t dt$ given $\Delta^2 \mu_x$, $\Delta^2 \mu_y$, $\Delta^2 \mu_z$ and $\Delta^2 \int g_x dt$, $\Delta^2 \int g_y dt$, $\Delta^2 \int g_z dt$ the latter being generated elsewhere. The linear drifts are corrected by initial settings of the proper "delta" registers of the QDPU. A conventional DDA requires a word time to compute drift corrections typically accomplished in a quantization integrator. The two scaling operations do the work of two additional DDA integrators. One of the cross product terms of the velocity computation is also computed in the same QDPU (see the $V_z \Delta \int \omega_g dt$ transfer in Figure 11-1). Thus, the QDPU of the diagram does the processing of 5 DDA integrators. As a result of there being involved in the sub-routine only 2 R registers instead of 5 R registers as in the conventional DDA, and as a result of multi-increment computation accuracy instead of single increment, the accuracy of computation of the QDPU is greatly increased. Since 3 of the 6 cross product terms are generated in the QDPU of the type in Figure 11-1, the remaining 3 must be computed elsewhere. By selecting 3 QDPU, programmed to carry out computations involving other V_x, y, z or $\int \omega_x, y, z dt$ terms the computation of all μ terms in such parallel assignment can amount to a total of less than 1 QDPU. In summary, it is seen that velocity computation and generation scaled-drift corrected inputs and outputs which require 18 integrators in a conventional DDA are executed in the QDDA with 3.75 QDPU, the QDDA, in effect, executing the





Note: $\Delta^2 p_x$ computed in flight path angle calculation

3 WT (instead of 15 WT for Conventional DDA)

Figure 11-2. Missile Velocity Calculation with Scale Factor: Linear Drift and Bias Error Correction of Inputs and Output of the Digital Computer

computation 4.8 times faster than the conventional DDA. The flow diagram of the routine is presented in Figure 11-2.

C. Computation of Flight Path Angle, Velocity Relative to the Fixed Pseudo Coordinate System, Radial Distance and Lift-Drag Ratio Computation for Re-entry

1. Routines Investigated - One of the critical quantities that must be known upon starting re-entry is the flight path angle. The total velocity relative to the fixed pseudo coordinate system is determined by

$$V = \sqrt{v_x^2 + v_y^2 + v_z^2} \quad (\text{XI-15})$$

These two calculations may be executed with high efficiency in the parallel-multi-increment QDDA making use of the $\sqrt{x^2 + y^2}$ computation capability of one of the two parallel channels. Thus, the intermediate quantity u given by

$$u = \sqrt{v_x^2 + v_y^2} \quad (\text{XI-16})$$

is used in both the velocity computation given by

$$V = \sqrt{v_z^2 + u^2} \quad (\text{XI-17})$$

as well as flight path angle computation.

The lift-drag ratio is another important quantity which can be determined by either pure inertial means or by air data measurements. The inertial method depends upon measurements from body mounted accelerometers and calculation of the true angle of attack. The equations required are as follows:

$$L/D = \frac{A_x \sin \alpha_t - A_z \cos \alpha_t}{-A_x \cos \alpha_t - A_z \sin \alpha_t} \quad (\text{assumes } 0 \text{ angle of bank}) \quad (\text{XI-1c})$$

where

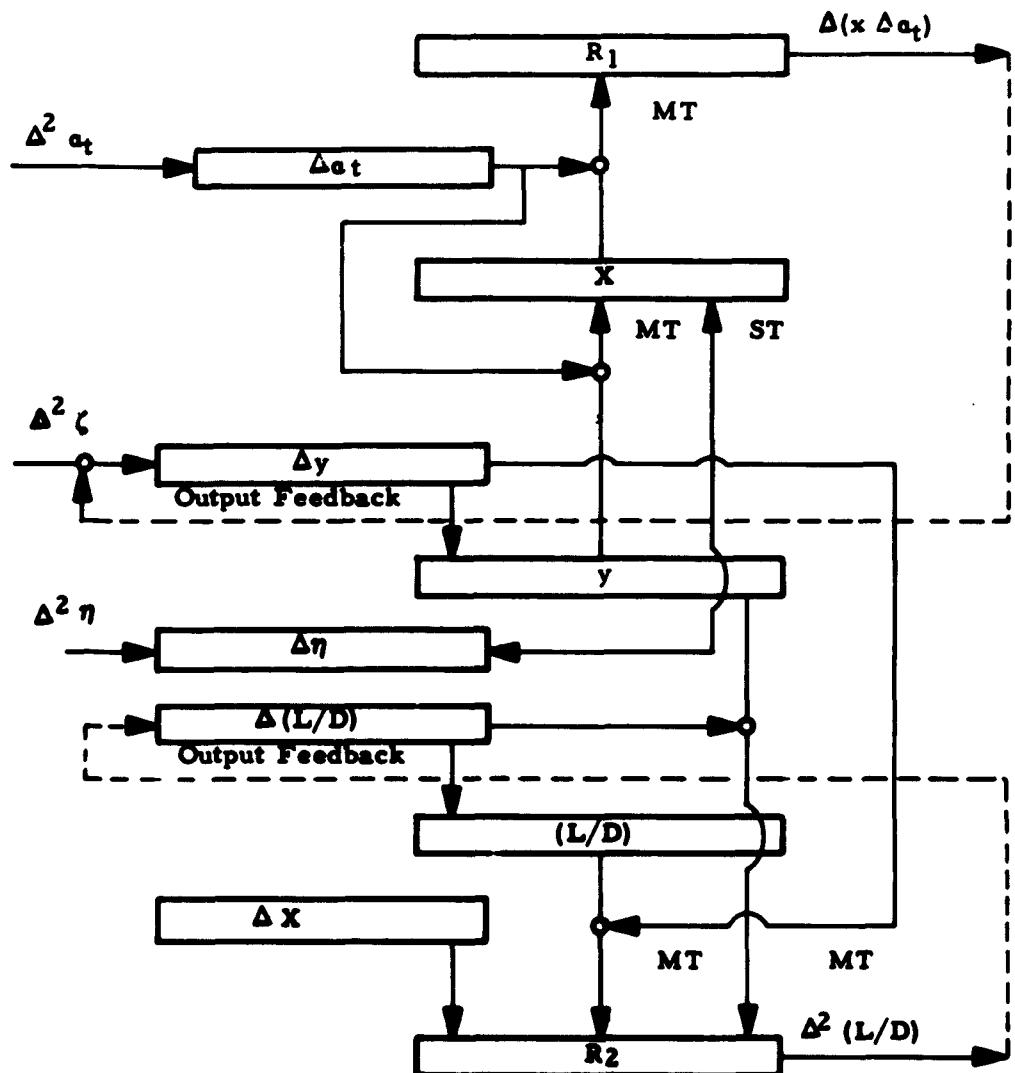
$$\alpha_t = \theta - \gamma \quad (\theta = \text{elevation angle as read from the inertial platform} \\ \gamma = \text{flight path angle computed in previous routine})$$

The A_x , A_z are whole word inputs to the digital computer. QDD³A inputs may alternatively be mechanized as A_x , A_z directly or dA_x , dA_z the latter formed by special input differencing

2. Program for Mechanization With Double Integration
Mode - Intermediate phase investigations of programs for sub-routines for mechanization with a double integration mode were carried out for the pertinent calculations to evaluate the special mode as well as general versatility of the QDD³A. While portions of the program assuming the double integration mode do not involve it, they are presented together because in the later studies in which the special mode was discarded (on later evaluation of merits and demerits) they are most efficiently programmed (and when the double precision mode was developed more precisely computed) as a single coupled routine which may be compared with results presented here. Assuming the double integration mode, a program for the calculations is presented in diagrams on the following two pages with a detailed QDPU mode diagram presented on the third page. The routine which requires a total of 29 DDA integrators (in 29 word times) in a conventional DDA is

PROGRAM I

A QDPU
MODAL PROCESSING FOR THE
LIFT-DRAG COMPUTATION



6 Integrator Equivalent
in 1 Word Time

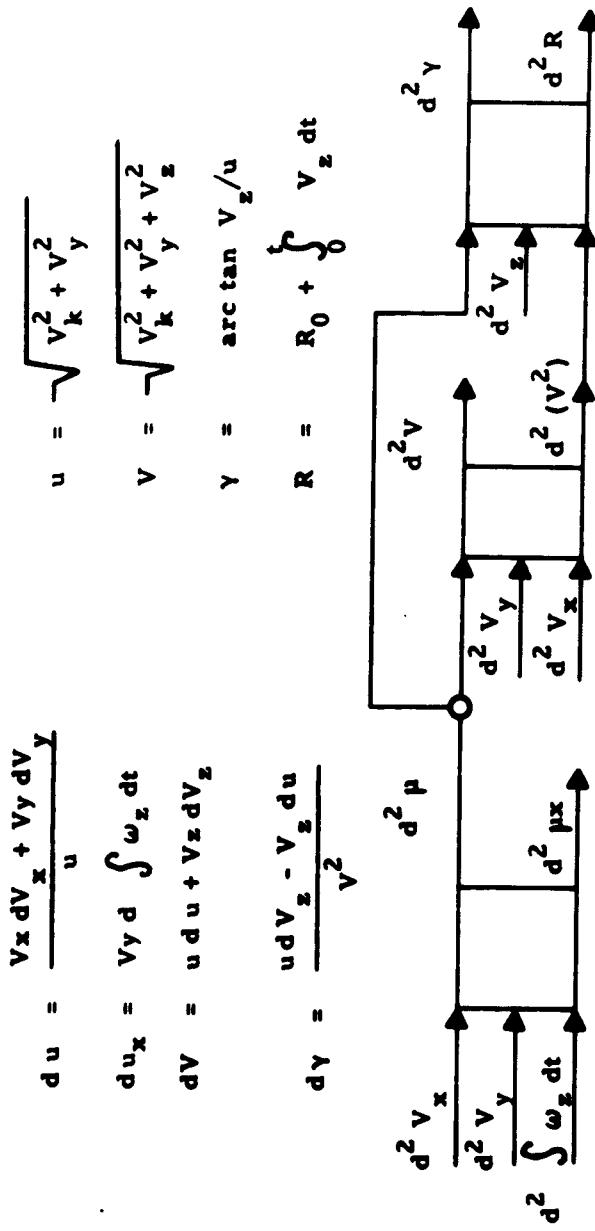
executed by 6 QDPU (in 6 word times) implying 1 QDPU :
4 5/6 DDA integrators.

3. Program for Mechanization Without Double Integration Modes - The QDPU program calculations and schematics for mechanizations without double integration mode are designated Program II and presented on the two pages following Program I. It is seen that the work of 28 DDA integrators is accomplished by 6 QDPU without double integration mode hence 1 QDPU : 4 2/3 DDA integrators. This result and other subroutine analyses as well as two other important considerations, led to the decision to discard the double integration mode.

The sinusoid computation is one of the most error sensitive of important DDA computations. With correct integration algorithm the major limitation in accuracy stems from roundoff error. Assuming 3 bit increment computation, the level of accuracy attainable in sinusoid at intermediate iteration rates on external input angles is certainly an order of magnitude higher than attainable with a conventional DDA with correct integration algorithm. Computation capability analysis places the improved performance, however, as still one of the limits in overall system performance. On development of the double precision mode feature, it was seen that this limitation could be removed without cost in integration rate. This assumes that double integration mode is mechanized, since the sinusoid requires 1 word time, at least, in a two output QDPU without double integration capability. The final determining factor in the decision to not mechanize a double integration mode is that the word time of the QDPU

PROGRAM I PART A

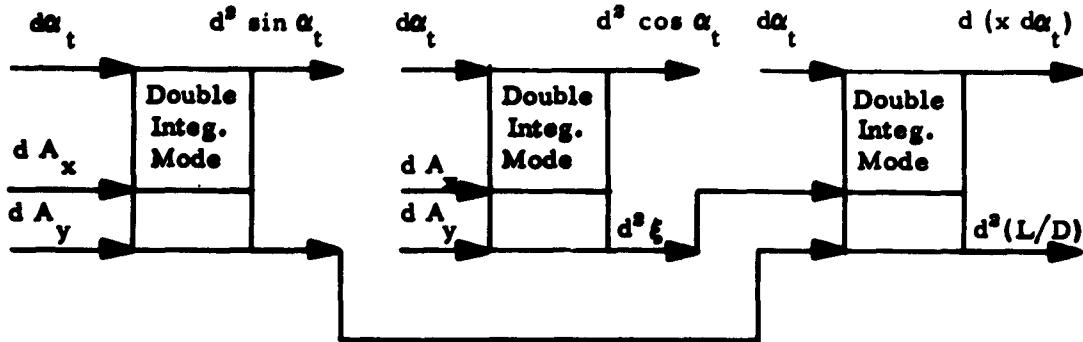
**MECHANIZATION WITH DOUBLE INTEGRATION MODE (ASSUMED)
COMPUTATION OF FLIGHT PATH ANGLE, VELOCITY RELATIVE
THE FIXED PSEUDO COORDINATE SYSTEM AND
RADIAL DISTANCE (MULTI-INCREMENT)**



3 WT (Instead of 17 WT by Conventional DDA)

Note: ux is one of the intermediate quantities in the velocity vector computation described in the preceding section.

Program I Part B Mechanization With Double Integration (Assumed)
Multi-QDDA Computation of
Lift-Drag Ratio For Re-entry
Body Mounted Accelerometer Inputs



3QDPU = 12 DDA Integrators

$$L/D = \frac{A_x \sin \alpha_t - A_z \cos \alpha_t}{-A_x \cos \alpha_t - A_z \sin \alpha_t}$$

is accomplished by multi-increment QDDA by calculation sets:

$$d\alpha_t = (d\theta - dy) \quad (\text{XI-19})$$

$$d\xi = -\cos \alpha_t dA_x - \sin \alpha_t dA_z; \quad d\eta = \sin \alpha_t dA_x - \cos \alpha_t dA_z; \quad d(L/D) = \frac{d_x - (L/D) dy}{y} \quad (\text{XI-20})$$

$$\begin{cases} d \sin \alpha_t = \cos \alpha_t d\alpha_t \\ d \cos \alpha_t = -\sin \alpha_t d\alpha_t \end{cases}$$

$$\begin{cases} d \sin \alpha_t = \cos \alpha_t d\alpha_t \\ d \cos \alpha_t = -\sin \alpha_t d\alpha_t \end{cases}$$

$$\begin{cases} dy = x d\alpha_t + d\xi \\ dx = -y d\alpha_t + d\eta \end{cases}$$

where x, y are defined as

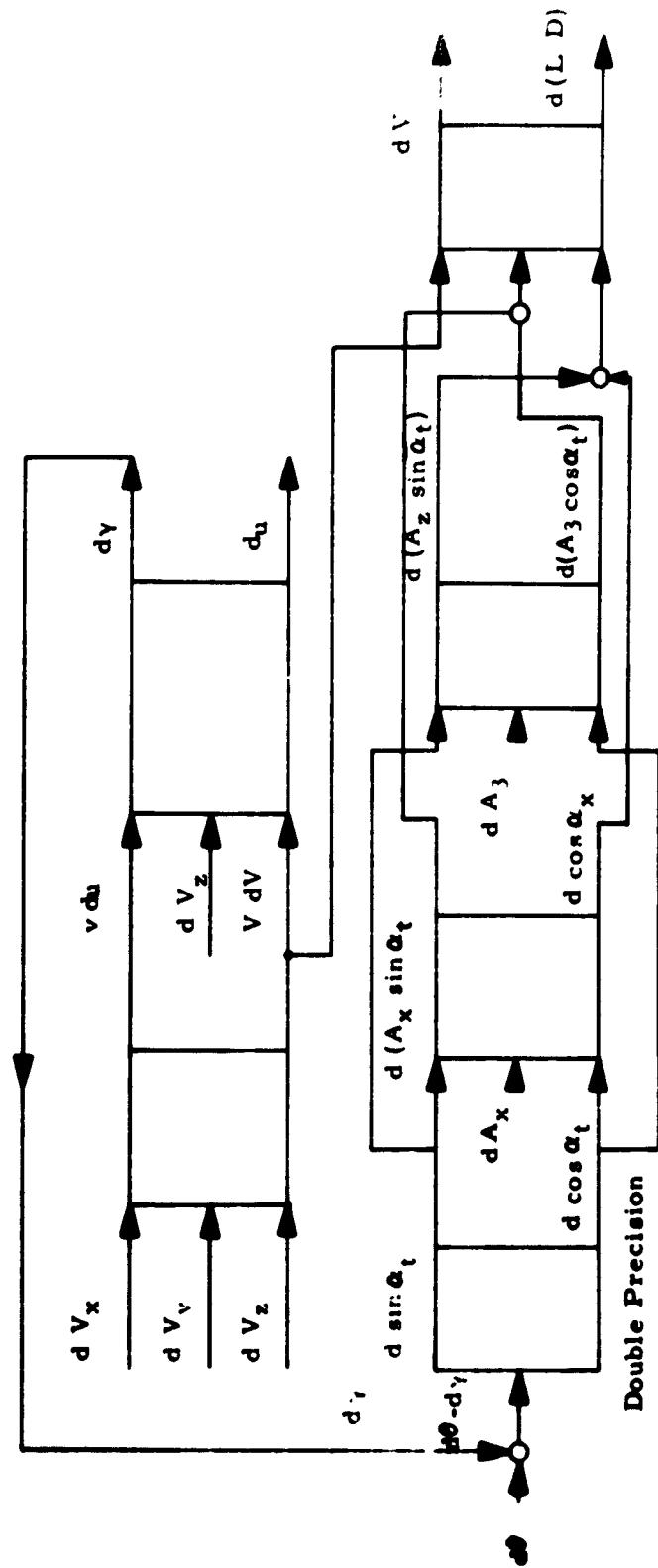
$$x = A_x \sin \alpha_t - A_z \cos \alpha_t$$

$$y = -A_x \cos \alpha_t - A_z \sin \alpha_t$$

PROGRAM II

**MECHANIZATION WITHOUT DOUBLE INTEGRATION MODE
RE-ENTRY CALCULATIONS: FLIGHT PATH ANGLE,
TOTAL VELOCITY, LIFT DRAG RATIO
(MULTI-INCREMENT)**

The sinusoid is computed with double transfer bit length (amounting to double precision)



Program II Mechanization Without Double Integration Mode

Re-entry Calculations: Flight Path Angle,

Total Velocity, Lift - Drag Ratio

(Multi-Increment, With Double Precision Mode)

$$\begin{aligned}
 [udu] &= V_x dV_x + V_y dV_y \\
 [vdv] &= V_z dV_z + V_x dV_x + V_y dV_y \\
 du &= \frac{[udu]}{u} \\
 dy &= \frac{udV_z - V_z du}{V^2} \\
 d \sin \alpha_t &= \cos \alpha_t d\alpha_t \\
 d \cos \alpha_t &= -\sin \alpha_t d\alpha_t \\
 d(A_x \sin \alpha_t) &= \sin \alpha_t dA_x + A_x d \sin \alpha_t \\
 d(A_x \cos \alpha_t) &= \cos \alpha_t dA_x + A_x d \cos \alpha_t \\
 d(A_z \sin \alpha_t) &= \sin \alpha_t dA_z + A_z d \sin \alpha_t \\
 d(A_z \cos \alpha_t) &= \cos \alpha_t dA_z + A_z d \cos \alpha_t \\
 d(L/D) &= \frac{dL^* - (L/D) dD^*}{D^*} \\
 dv &= \frac{[vdv]}{V}
 \end{aligned}
 \quad \left. \begin{array}{l} 5 \\ 6 \\ 7 \\ 8 \\ 2 \\ 4 \\ 4 \\ 4 \\ 7 \end{array} \right\}$$

28

1 QDPV = $4^{2/3}$ DDA Integrators

(which is programmable) must be increased 50 to 100% over the word length determining desired resolution. Thus the double precision mode, which requires only 10 to 15% longer word, actually equals the net performance counting integrator equivalent and iteration rate together, without further elaboration of the computer.

4. Geographic Coordinate Computation for Aerospace Navigation and Evaluation: Sinusoid Computation Modes for Alternative QDPU Designs

a) Non-Polar Flight

The primary navigation for the aerospace mission is done in a pseudo-coordinate system for compatibility of the outputs with desired orbital display quantities. Some of the terms, however, such as gravitational effects are functions of the geometric coordinates. In addition, during the re-entry and landing phases it is most convenient to calculate "geographic coordinates". The "pseudo" coordinates make for ease of computation in many of the navigation routines other than the coordinate conversion. The computation involved in coordinate conversion where flight very near the poles is excluded may be discussed in terms of the explicit form:

$$\theta = \text{arc sin} [\cos \phi \sin \theta \sin B + \sin \phi \cos B] \quad (\text{XI-21})$$

$$= \lambda_u - \text{arc cos} \left[\frac{\cos \phi \cos \theta}{\cos \phi} \right]$$

θ = geocentric longitude

λ = geocentric longitude

$$\phi_g = \phi + \epsilon \sin 2\theta$$

$$\lambda_g = \lambda + \int_0^t \Omega dt$$

ϕ_g = geographic latitude

λ_g = geographic longitude

The given $d\theta$, $d\psi$, $\sin \psi$, $\cos \psi$ more general case for flight through the poles will be discussed after the simpler computation programming problem is delineated and QDD²A programming performance stated in the next set of schematics.

5. Complete Geographic Coordinate Calculations of the Proposed QDPUs (Multi-Increment, Without Double Integration Mode) - Figure 11-3 shows the geographic coordinates of the following calculations.

Calculations: $\phi = \arcsin [\cos \psi \sin \theta \sin \beta_0 + \sin \psi \cos \beta_0]$ (XI-22)

$$\lambda = \lambda_n - \arccos \left[\frac{\cos \phi \cos \theta}{\cos \psi} \right]$$

$$\phi_g = \phi + \epsilon \sin 2\theta$$

$$\lambda_g = \lambda + \int_0^t \Omega dt \text{ where } \phi_g, \lambda_g \text{ = geographic coordinates}$$

Inputs: $d^2\theta$, $d^2 \cos \psi$, $d^2 \sin \psi$ (XI-23)

Differential Relations used in each QDPU and DDA integrator equivalents:

$$\frac{d\phi}{d\phi} = \frac{\sin \beta_0 d(\cos \phi \sin \theta) + \cos \beta_0 d \sin \phi}{\cos \phi} \quad \left. \begin{array}{l} \\ \end{array} \right\} 5 \text{ DDA Integrators}$$

$$d \sin \phi = \sin \beta_0 d(\cos \phi \sin \theta) + \cos \beta_0 d \sin \phi \quad \left. \begin{array}{l} \\ \end{array} \right\}$$

$$d(2^e \epsilon \cos 2\phi) = \frac{\cos \phi d \sin \phi + \sin \phi d \cos \phi}{\epsilon^{-1} 2^{-e}} \quad \left. \begin{array}{l} \\ \end{array} \right\} 4 \text{ DDA Integrators}$$

$$d \cos \phi = -\sin \phi d \phi \quad \left. \begin{array}{l} \\ \end{array} \right\}$$

$$d(\cos \phi \cos \theta) = \cos \phi d \cos \theta + \cos \theta d \cos \phi \quad \left. \begin{array}{l} \\ \end{array} \right\} 4 \text{ DDA Integrators}$$

$$d(\cos \phi \sin \theta) = \cos \phi d \sin \theta + \sin \theta d \cos \phi \quad \left. \begin{array}{l} \\ \end{array} \right\}$$

$$d \cos \theta = -\sin \theta d \theta \quad \left. \begin{array}{l} \\ \end{array} \right\} 2 \text{ DDA Integrators}$$

$$d \sin \theta = \cos \theta d \theta \quad \left. \begin{array}{l} \\ \end{array} \right\}$$

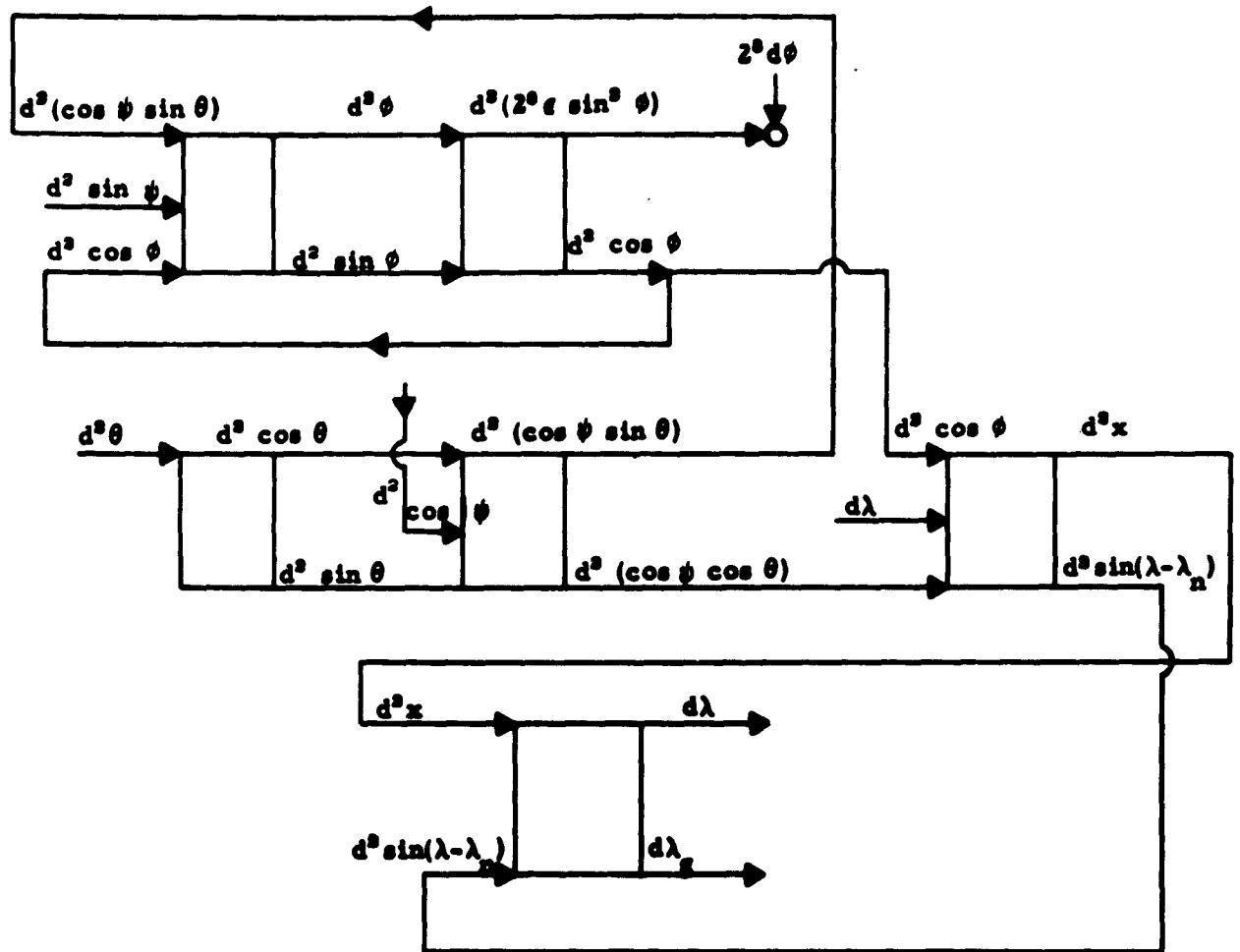
$$d \sin(\lambda - \lambda_n) = x d\lambda \quad \left. \begin{array}{l} \\ \end{array} \right\} 5 \text{ DDA Integrators}$$

$$\frac{dx}{dx} = \frac{d(\cos x \cos \theta) - x d \cos \phi}{\cos \phi} \quad \left. \begin{array}{l} \\ \end{array} \right\}$$

$$\frac{d\lambda}{d\lambda} = -\frac{dx}{\sin(\lambda - \lambda_n)} \quad \left. \begin{array}{l} \\ \end{array} \right\} 4 \text{ DDA Integrators}$$

$$d\lambda_g = d\lambda + \int_0^t \Omega dt \quad \left. \begin{array}{l} \\ \end{array} \right\}$$

6 QDPU = 24 DDA Integrators



6 QDPU \approx 24 DDA Integrators

Figure 11-3. Complete Geographic Coordinate Calculations by the Proposed QDPU Multi-Increment, Without Double Integration Mode

6. Geographic Coordinates Calculation by the QDPU (Multi-Increment Assuming a Double Integration Mode Not Incorporated in the Final Design) - Figure 11-4 shows the geographic coordinates of the following calculations.

Calculations:

$$\phi = \text{arc sin} (\cos \psi \sin \theta \sin \beta_0 + \sin \psi \cos \beta_0) \quad (\text{XI-24})$$

$$\lambda = \lambda_n - \text{arc sin} \left[\frac{\cos \psi \cos \theta}{\cos \phi} \right] \quad (\text{XI-25})$$

QDPU Operations (Expressed in single increments):

$$\begin{aligned} \text{I} & \left\{ \begin{array}{l} d \cos \phi = d \int \int \cos \phi d \phi \\ d(\cos \psi \sin \theta) = \sin \theta d \cos \psi + \cos \psi d \sin \theta \end{array} \right. \\ \text{II} & \left\{ \begin{array}{l} d \sin \theta = d \int \int \sin \theta d \theta \\ (d(\cos \psi \cos \theta)) = \cos \theta d \cos \psi + \cos \psi d \cos \theta \end{array} \right. \\ \text{III} & \left\{ \begin{array}{l} d\phi = \frac{\sin \beta_0 d(\sin \phi / \sin \beta_0)}{\cos \phi} \\ d \frac{\cos \psi \cos \theta}{\cos \phi} = \frac{d(\cos \psi \cos \theta)}{\cos \phi} - \left(\frac{\cos \psi \cos \theta}{\cos \phi} \right) d \cos \phi \end{array} \right. \\ \text{IV} & \left\{ \begin{array}{l} d \sin(\lambda - \lambda_n) = d \int \int \sin(\lambda - \lambda_n) d \lambda \\ d\lambda = \frac{-d \left(\frac{\cos \psi \cos \theta}{\cos \phi} \right)}{\sin(\lambda - \lambda_n)} \end{array} \right. \end{aligned}$$

$$\text{Calculations: } \phi = \arcsin [\cos \psi \sin \theta \sin \beta_0 + \sin \psi \cos \beta_0] \quad (\text{XI-26})$$

$$\lambda = \lambda_n - \arccos \left[\frac{\cos \psi \cos \theta}{\cos \theta} \right] \quad (\text{XI-27})$$

where $\theta_g = \theta + e \sin 2\phi$

$$\lambda_g = \lambda + \int_0^t \Omega dt, \theta_g, \lambda_g = \text{geographic coordinates}$$

Inputs: $d^2\theta, d^2 \cos \psi, \cos \beta_0, d^2 \sin \psi$ (XI-28)

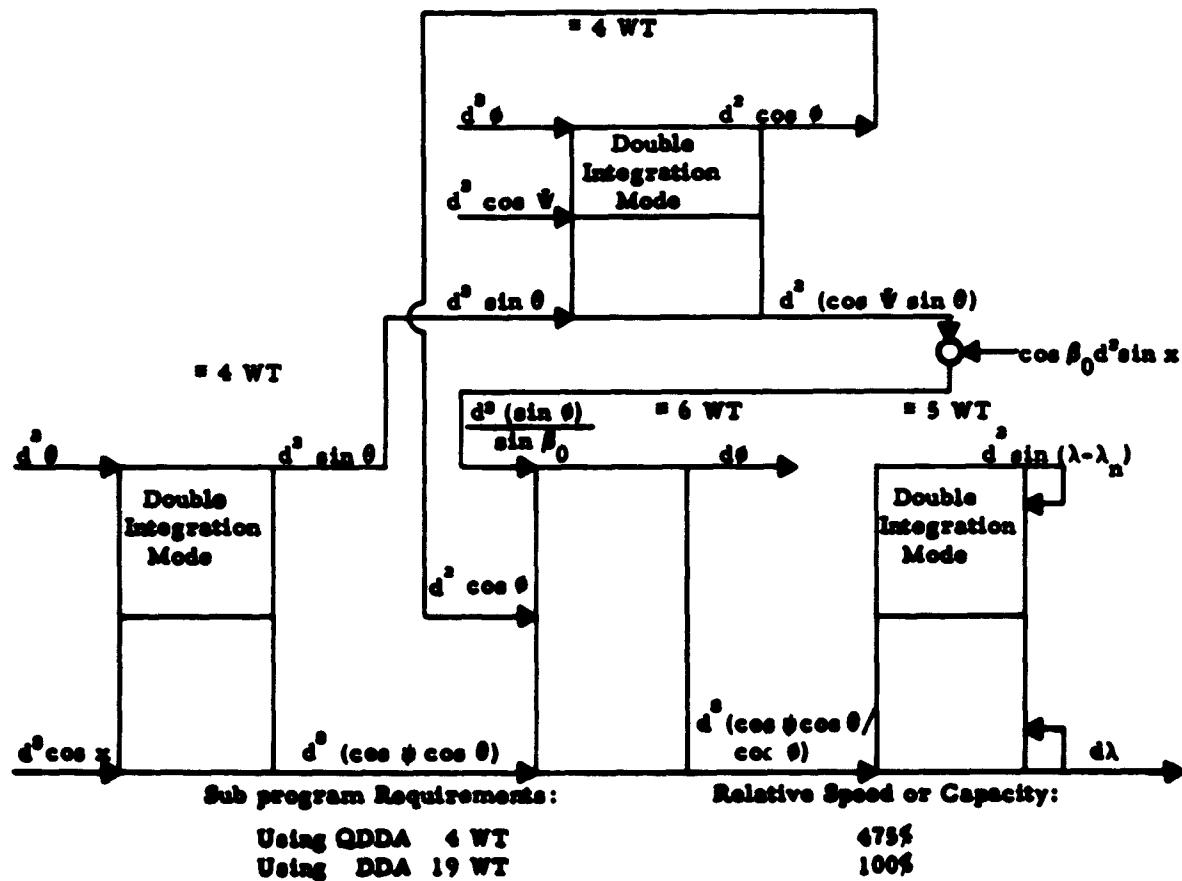


Figure 11-6. Geographic Coordinates Calculation By The QDPU (Multi-Increment and Assuming A Double Integration Mode Not Incorporating in Final Design)

D. Flight Over the Poles and the Longitude Discontinuity - One of the most strongly felt factors in the application of incremental computers in airborne and aerospace applications is the computation problems in handling variables with isolated discontinuities. In a number of recent system developments within the field, including a full aerospace mission computer with GP-DDA, the impress of this factor has led to highly discriminated computation allocation to the DDA and such demanding assignment to the GP section that only new hardware developments and increases in GP computer complexity, already unwieldy, presented hope of approximating computation requirements along that design path. The basic reason for this impact of computer complexity stems from the basic speed disadvantage of the GP relative to the DDA as discussed in paragraph 11.4D which presents and analyzes the energy management computation program. That the allocation to other than DDA is not inherent for all computations with isolated discontinuities, is shown in paragraph 11.4G where the important problem of doppler damping is shown soluble with required accuracy by the QDD²A. That the allocation to other than DDA computations for flight over the pole, and generally across a longitude discontinuity is not inherent, will be discussed here. It should be noted that most isolated discontinuities in real time computation present themselves not so much as a result of inherent needs of computation, but as a result of arbitrary choice of coordinates which are to be displayed, as for example, geographic coordinates. While it is true that no fixed coordinate system can define path and motion on a closed surface without discontinuities, a non-fixed or conditionally varying coordinate system defining the path and motion can be free of important discontinuities. Thus, a coordinate system which is identical to geographic coordinates

in regions further than 30 miles from a pole, and conditionally varied within regions close to the pole, can be free of important discontinuities. Such a coordinate system can serve as a practical geographic coordinate system since actually the value of precise coordinates is primarily away from the pole. Consider the problem of the longitude discontinuity at $+180^\circ - 180^\circ$ for all latitudes; here the DDA would be required to compute longitude which changes 360° almost instantaneously. The supervision of this change in a DDA by GP is costly in programming; therefore, a DDA capable of making this change itself without extensive programming has a real advantage over any existing DDA. It is proposed that this capability be implemented by a simple mechanization elaboration (not included in decision modes of Chapter XII because of its recent derivation). A new decision mode capable of accomplishing this within the structure of a ternary DDA and the QDD²A has the output of sign and an amplitude bit communicated in the natural manner of an arithmetic output. Here, the integrator or QDPU in this decision mode puts out two bits of information, a Y register sign and most significant bit rather than normal output according to natural overflow or general (quotient) output criterion. The output in this mode requires no arithmetic process for output, but simply reads an already present register value and does not itself disturb the pertinent Y register value at that iteration. Rather, the output is a decision command output programmed to be an output to the pertinent and any other desired integrator or QDPU. The decision command conveys that a step change of a given sign and fixed magnitude is to be made in the receiving y register. Thus, the longitude variable in a y register puts out sign of longitude and an amplitude bit (1 if positive

or 0 if negative) when the magnitude exceeds 180° . Since flight is continuous, the excess over 180° produced by the last Δy becomes the correct change had the 360° step been instantaneously subtracted at that iteration. Having programmed accordingly, the output is an input additional to the longitude increment to the same y register. At the next iteration after discontinuity crossing the decision response acts to subtract or add 1 from the y register, mechanization for which is available already by modal action of an unused single transfer unit ordinarily required in general (quotient) algorithm computation. Next, consider the more challenging design and/or programming problem of geographic coordinate computation in which the poles are closely or directly passed and where long term accuracy after passing the pole is the prime consideration. Preliminary analysis indicating that a combination design modification and special programming of the problem appears promising. In this approach which may be called the moving pole method, the coordinate system is effectively modified only when $|\phi| > 90^\circ - \epsilon$ where $\epsilon = 1/2^\circ$. The approach appears particularly simple for orbital flight where the approximate minimum pass distance from the pole is essentially known. Upon entering the pertinent polar region, as detected by a decision operation in the QDD²A, the modified coordinates are formed by incremental shifting of the pole from that of true geographic position through making increments of orbital inclination B in the formulae stated in the preceding section. For example, coordinates may be selected to have magnitude equal to ordinarily computed increments of pseudo longitude, pseudo latitude and longitude coordinates are orbital coordinates (perfect flight has zero pseudo latitude). Thus, as

Best Available Copy

the missile approaches the true pole, the vertical pole of the modified coordinates shifts away such that the minimum distance may be shown to be $\epsilon/\sqrt{2}$, and as the missile leaves the region then the virtual pole returns to the exact position of the true pole. This proposed computation method assures long term accuracy for orbital flight passing repeatedly over or near the poles. The decision modes of a conventional DDA and their counterpart in the QDD²A appear to provide adequate basis for supervising the moving pole method, but with considerable programming. The goal of developing an incremental computer capable of performing complete aerospace guidance and control functions now assigned to a GP-DDA system (see Chapter XIV) implies that provision should be made to effect highly versatile self-supervision without appreciable increase of total computation time. Thus the mechanization of a new decision command mode is proposed, namely, one which generates a bit output, one bit being the sign of y in a y register. The other is a "1" only when $|y|<2^{-K}$. Mechanization of the arithmetic part of this mode (which is a generalization of the decision command mode part for computation of longitude across the discontinuity) is straightforward. The cost of making K programmable has not been evaluated. Here, the quantity $\sin \theta$ is programmed as y and the decision command output is sent to a QDPU where it acts as positive or negative full rate when $|y|<2^{-K}$, in the computation of $\cos \beta$, $\sin \beta$. Ideally, the net change of β in the whole process is zero. This is only approximately assured provided the missile has constant velocity over the 1° region. Thus, probably a more sophisticated choice of functions than $\sin \theta$ is indicated. Another method involving direct count down of all changes of β should be examined.

1. Gravity, Angular and Angular Rates Computations, for Aerospace Flight - The gravity computation for aerospace flight requires precision division computation. The gravity computations, angular and angular rates, are determined from the relations:

$$\omega_x = -\frac{V_y}{R} \quad (\text{XI-29})$$

$$\omega_y = \frac{V_x}{R} \quad (\text{XI-30})$$

$$\omega_z = \dot{\sigma} \sin x \quad (\text{XI-31})$$

$$\dot{x} = -\omega_x = \frac{V_y}{R} \quad (\text{XI-32})$$

$$\dot{\theta} = \frac{V_x}{R \cos x} \quad (\text{XI-33})$$

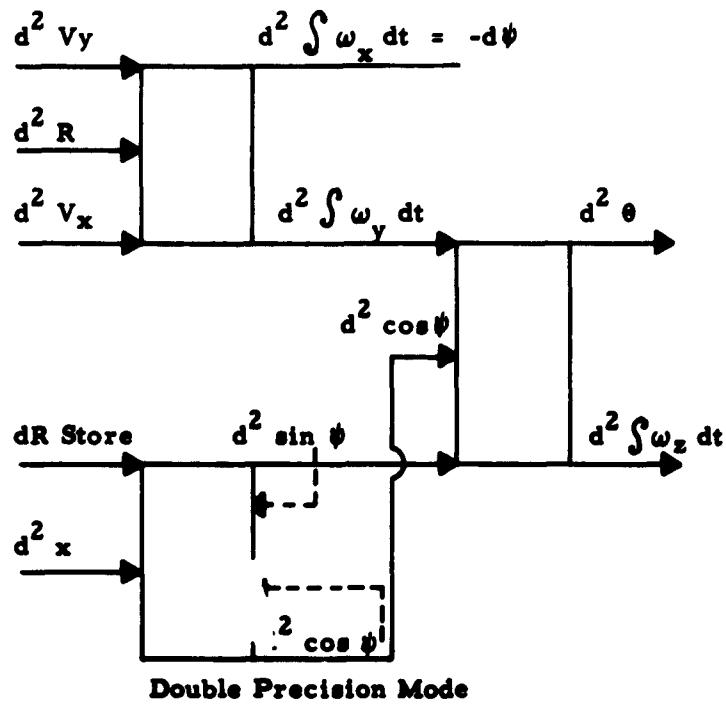
$$dR = dV_z \quad (\text{XI-34})$$

$$g_{x0} = 0.485 g_e \left(\frac{a}{R} \right)^4 \epsilon \sin 2\theta \cos B_o \quad (\text{XI-35})$$

$$g_{y0} = 0.485 g_e \left(\frac{a}{R} \right)^4 \epsilon \sin 2\theta \sin B_o \quad (\text{XI-36})$$

$$g_{z0} = g_e \left(\frac{a}{R} \right)^2 \left[1 - 1.454 \left(\frac{a}{R} \right)^2 \epsilon \sin \phi \right] \quad (\text{XI-37})$$

The angular and angular rate calculation QDD³ A program and QDPU integrator equipment (for θ , ϵ , $d\omega_z$) is summarized on the following page and illustrated in Figure 11-5.



3QDPU = 11DDA Integrators of Programming

Figure 11-5. Angular Rate Computation Program for the QDDA

$$\left. \begin{aligned} d\int \omega_x dt &= -d = -d \int \frac{v_y}{R} dt \\ d\int \omega_x dt &= d \int \frac{v_x}{R} dt \end{aligned} \right\} 1QDPU = 4DDA Integrators \quad (XI-38)$$

$$\left. \begin{aligned} d\int \omega_x dt &= d \int \frac{\sin \phi d \int \omega_y dt}{\cos \phi} \\ d\sigma &= d \int \frac{d \int \omega_y dt}{\cos \phi} \end{aligned} \right\} 1QDPU = 5DDA Integrators \quad (XI-39)$$

$$\left. \begin{array}{l} d \cos \psi = -\sin \psi d\phi \\ d \sin \psi = \cos \psi d\phi \\ dR \text{ store} \end{array} \right\} \quad \begin{array}{l} 1QDPU \equiv 2DDA \text{ Integrators} \\ \text{Double} \\ \text{Precision} \end{array} \quad (\text{XI-40})$$

The gravity computations are simplified by introducing the variables w, v defined by

$$w = 1.454 \epsilon v^2 \sin^2 \phi$$

$$v = g_e \left(\frac{a}{R} \right)^2$$

then it may be shown

$$dg_{x_0} = \frac{2g_{x_0} dv}{v} + (K_{z_0} v^2 - 2w) d\phi \quad (\text{XI-41})$$

$$dg_{y_0} = \tan B_0 dg_{x_0} \quad (\text{XI-42})$$

$$dg_{z_0} = dv - dt \quad (\text{XI-43})$$

$$\text{where } K_{x_0} = \frac{2(1.485) \epsilon \cos B_0}{g_e}, \quad K_{z_0} = \frac{1.454 \epsilon}{g_e}$$

The calculation may be executed in the QDDA by programming the calculations:

$$\left. \begin{array}{l} da = \frac{g_{x_0} (K_{x_0} d\phi)}{(K_{x_0}^2 / 2K_{z_0})} \\ d\hat{a} = \epsilon (K_{x_0} d\phi) \end{array} \right\} \quad 1QDPU \equiv 3DDA \text{ Integrators} \quad (\text{XI-44})$$

$$\left. \begin{array}{l} da = \frac{2\omega dv}{v} \\ d\delta = \frac{2g_x dv}{v} \end{array} \right\} \quad IQDPU \approx 6DDA \text{ Integrators} \quad (XI-45)$$

$$\left. \begin{array}{l} d\sigma = -\frac{2\sigma dR}{R} \\ de = vdv - (2/b_{z0}) dw \end{array} \right\} \quad IQDPU \approx 6DDA \text{ Integrators} \quad (XI-46)$$

A schematic showing the programming of these gravity calculations is shown in Figure 11-6. The gravity, angular and angular rate, programs combined indicate an average integrator equivalent of the QDPU of 4-1/3 including one double precision mode.

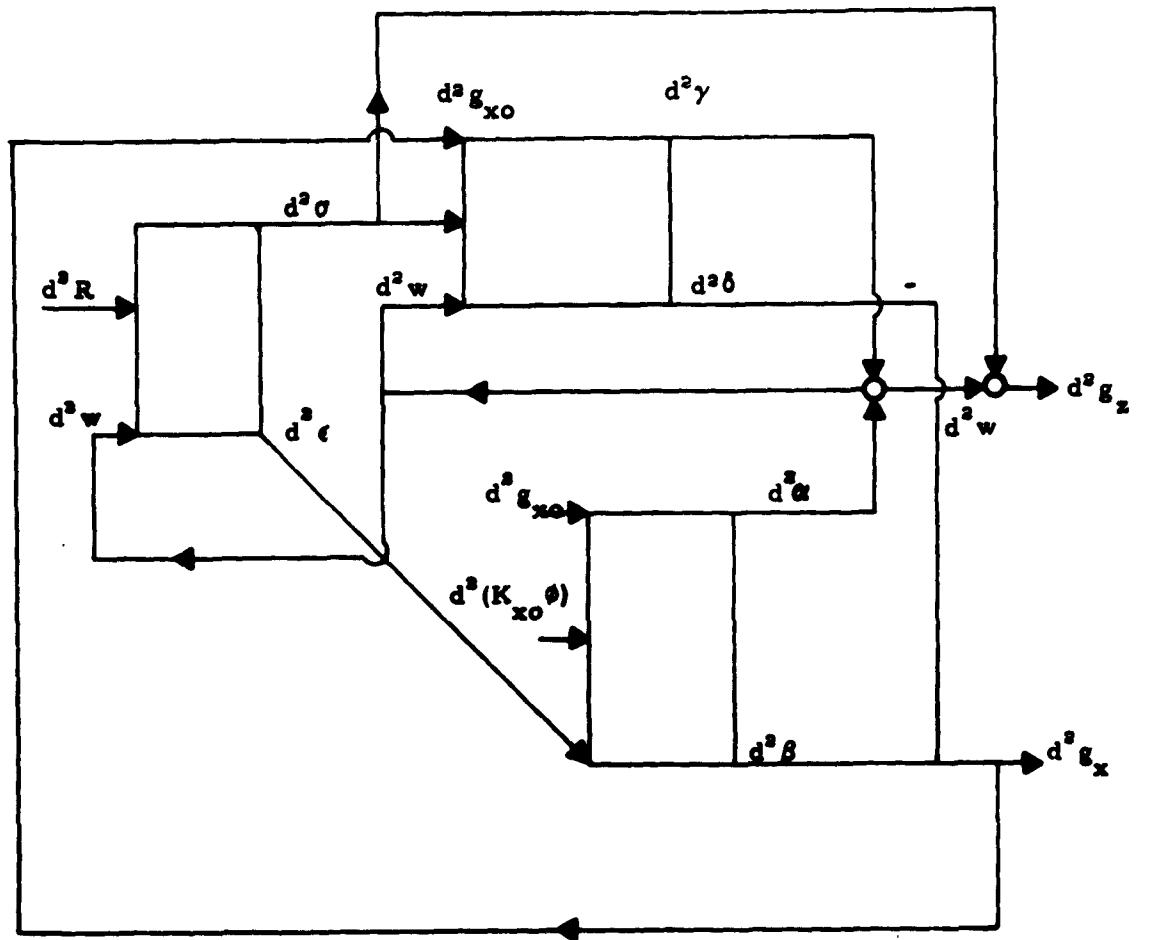


Figure 11-6. Gravity Computation Program for the QDDA

($d^2 g_y = \tan B_0 d^2 g_x$ and $d^2 (K_{x_0} \theta) = K_{x_0} d^2 \theta$ done elsewhere
and not counted here for QDDA or DDA)

3QDPU = 15 DDA Integrators

11-2. THRUST CUT-OFF CALCULATIONS BY THE QDD²A, AND THE CONVENTIONAL COMPUTATIONS ALLOCATION APPROACH AND LIMITATIONS - Perhaps one of the most crucial computer operations is control of the rocket thrust termination. One previous analysis concluded a general purpose computer was necessary for the allocation with full time devoted to achieve accurate cut-off during the period when cut-off is imminent.* The cut-off condition is D such that $|D| < D_0$, where $D_0 = 0.5 \text{ ft/sec}$ say, and

$$D = K_1 V_{E_x} + K_2 V_{E_y} + K_3 V_{E_z} \quad (\text{XI-47})$$

where $V_{E_x} = V_{P_x} - V_x$

$$V_{E_y} = V_{P_y} - V_y$$

$$V_{E_z} = V_{P_z} - V_z$$

$V_P()$ = desired cut-off velocities

$V()$ = actual velocities

The quantities $K_1, 2, 3$ are updated at low rate. In order to limit the duration of high rate (1000 to 2000 iter/sec) cut-off calculation, which use essentially the full arithmetic capacity of the GP computer, a low rate calculation of the same form as above but with $D_0 = 30 \text{ ft/sec}$ has been used to turn on the high rate calculation only when cut-off is imminent. There are several implications of this conventional computation allocation approach for GP-DDA computer systems with relatively slow DDA for partial rather than full aerospace mission:

*A concessionary approach in which craft control and other functions are temporarily neglected.

- A. The quantities $V_{x, y, z}$ change primarily with inertial velocity increments (external inputs) during the cut-off decision period, however, the gravity and coriolis velocity changes internally generated at low rate have a granularity comparable to or greater than the required net accuracy. A similar effect of smaller magnitude holds for $V_{p_x, y, z}$.
- B. The interruption of the GP functions of auto-piloting, while the missile is under very high acceleration, is required during the cut-off decision making period implying the possibility of serious loss of thrust control at the most critical time.
- C. Any other functions, such as, telemetering or tracking in future aerospace programs would be interrupted at least momentarily.

11.3 SOURCES OF COMPUTATION IMPROVEMENT AND GENERAL QDD^{2A} PROPERTIES - Computer design and program allocation approaches which may remove these limitations are as follows: Reduced granularity of internally computed components of $(V_p - V)_{x, y, z}$ can be obtained with a higher speed DDA. By taking advantage of the continuity of the rate of change of these variables, extrapolation calculations by simple summation (in essence assuming constant rate of change) can provide accurate values during the cutoff decision period, even with low rate inputs to the high rate loop. For a single increment DDA, it would appear, that considerable additional programming would be required to generate a whole word or, at least a several bit word representation, of rate of change. A several bit increment DDA would not require the additional programming elaboration. The QDDA has natural cost free extrapolation capability in the high rate loop when updated by outputs of the low rate loop.

The interruption of computation functions, for other than cut-off computations, during the decision period is obviated by not doing the computations in the GP or (intermediate rate) internal incremental computation loop. The QDDA, in

input processing mode, can execute the cut-off decision calculations without any interruption whatsoever, and with at most an equivalent slow down of the computer of 10 to 15%, most of which buys input processing capability in addition to the thrust-cut-off operation.

11.4 PROGRAMMING THE QDDA FOR THRUST CUT-OFF DECISION CALCULATIONS

- A. Multi-Iteration Loop Set-Up - For a state of the art word rate, of 1.4×10^4 words/sec, the QDDA, designed to have 64 QDPU (equivalent to > 250 DDA integrators of program count), could have an iteration rate of 218 iter/sec, if no high rate input processing were programmed. If a high rate loop for thrust cut-off is performed at 1400 iter/sec, the QDDA has a set of write heads for 10 word lines. Two QDPU suffice for the thrust cut-off calculation. The indicated set-up would enable assignment of 4 other QDPU to other input processing functions leaving 59 QDPU for internal computations at 109 iter/sec. The program count for high rate input processing in aerospace computations in the case where there are no strap-down computations may not require more than 2 to 6 QDPU. If more input processing were required, the cost of another set of write heads and a small amount of logic would be imposed.
- B. Multi-Iteration Rate QDDA Computations for thrust cut-off. High iteration rate is required for thrust cut-off computations to avoid launch velocity error after thrust cut-off. High iteration rate is attainable at state of the art word rates only for very short computation routines. Properties of rates of change of the variables involved in thrust cut-off should be exploited to

limit the high rate computation program to acceptable size. In the QDDA, the thrust cut computations are executed in a relatively high rate input processing loop, in which other input processing functions may also be attained to the extent of high rate thrust cut-off program compatibility. The quantities K_1 , K_2 , K_3 are relatively slowly changing. The velocity deviation for thrust cut-off decision, D_o , is assumed constant. One of the K_i quantities say K_3 satisfies $K_3 >> 0$, hence consider the relation

$$\frac{D}{K_3} = \frac{K_1}{K_3} V_{E_x} + \frac{K_2}{K_3} V_{E_y} + V_{E_z} = K_1^* V_{E_x} + K_2^* V_{E_y} + V_{E_z} \quad (\text{XI-48})$$

as determining thrust cut-off according to the condition

$$\left| \frac{D}{K_3} \right| < \frac{D_o}{K_3} = D_o^* \quad (\text{XI-49})$$

The variation of D_o/K_3 in one slow iteration of the QDDA internal computation loop is negligible insofar as the significant variation in effective cut-off decision level which would result if the variation were neglected (instead of being 0.5 ft/sec it might be 0.501 ft/sec, a difference being far finer than necessary resolutions). With a view of eliminating computations of D (or D/K_3) in more than one place in the computer system, (to effect a program reduction), a very precise computation is sought. The thrust cut-off program switchings may then be held to the number of thrust stages of the mission. For high accuracy in a DDA computation of D (or D/K_3) the treatment of both K^* and V_E quantities as variables at the high rate have an advantage. Programming the D^* calculation in the QDDA will be outlined and shown to essentially obtain this advantage.

Each of 2 QDPU are programmed to execute one of the two parallel channels, a computation of form:

$$d(K^*V_E) = V_E dk^* + K^*dV_E \quad (\text{XI-50})$$

using in each QDPU two 6 registers and two y registers. The internal processing section computes $d^2 D^*$, $d^2 K^*$, $d^2 K_e^*$ at the intermediate rate (≈ 100 it/sec) and the QDPU in the high rate loop picks these up effectively at the intermediate rate (0 being communicated at intermediate intervals). A quantity dk^* is available with several bits resolution at high rate because of the low rate of change of the K^* . A 3 bit transfer QDDA mode computes the K^* quantities with full multi-increment accuracy at high rate. Consider a dV_E quantity which is composed of both high and low rate quantities,

$$dV_E = dV_{EL} + dV_I \quad (\text{XI-51})$$

where dV_I is accelerometer input at high rate and dV_{EL} includes computed gravity and earth rate induced changes at low rate. Computing dV_E and dV_{EL} at the high rate assures that V_{EL} may be multi-increment and that dV_{EL} be consistent with this scaling, when generated at low rate, in effect producing substantially the same outputs, as if generated at high rate. While actual computation at high rate would generate a dV_{EL} communication at intermediate phases of the long iteration interval, the lag free algorithm assumed in the low rate computation should make the alternate communication streams essentially equivalent. In conclusion, the computation of $d(K^*V_E)$ in the QDPU at high rate with the particular low rate communications is expected to have multi-increment accuracy at the high rate

so far as K^* and V_{E_L} are concerned. The accuracy effect of the external input V_I depends on a maximum accelerometer pulse rate and sensor, transducer accuracies. Pulse rates during the next 5 years are assumed under $10^4/\text{sec}$.

Having D or D^* available at high accuracy, the question of implementing decision for thrust cut-off in the best way is considered. At the 15g maximum thrust level the step size of D is about $1/3 \text{ ft/sec}$ at 1400 it/sec . If a decision for cut-off type is the conventional which tests for

$$|D| < D_o$$

then D could be chosen 0.25 ft/sec without chance of passing over the decision region. The rms error using this criterion is at least $0.25/\sqrt{3} = 0.15 \text{ ft/sec}$ assuming precise computation of D . This inherent error appears to be entirely consistent with state of the art system accuracy requirements. Note that the step size of dV_L computed at low rate is determined by the gravity magnitude of $1g$ and the low iteration rate. For several low iteration rates the step size of dV_L is:

IR	Max Step Size of V_L
15 it/sec	2.7 ft/sec
50 it/sec	0.64 ft/sec
100 it/sec	0.32 ft/sec

For 1 ft/sec error at launch as a maximum overall tolerance, it is concluded that dV_L should be computed in a single increment DDA at least at 70 to 100 ft/sec. This exceeds the iteration rate of a conventional DDA with the large program for a full aerospace mission assuming state of the art word rates. Three bit increment QDDA computation with 2nd difference communication could as a result of the high rate computation of dV_L compute d^2V_L (from which dV_L is obtained) at <15 ft/sec in internal computation and meet the accuracy requirements. Decision criterion for thrust cutoff is

$$|D| < D_0^*$$

with the conventional decision mode (which generates a decision command consisting of the sign of a quantity in y register). The thrust cut-off is determined by the logical product of two decision command signals generated by programming inputs to y₁ and y₂ registers in the DDA to form

$$y_1 = -D_0^* + D^* \quad (\text{XI-52})$$

$$y_2 = +D^* + D_0^* \quad (\text{XI-53})$$

Because the QDPU can be programmed for arithmetic and decision operations in parallel the two QDPU which compute D* can also generate the decision quantities for thrust cut-off. The program and operation of the 2QDPU used for thrust cut-off computations are presented in the schematic of Figure 11-7.

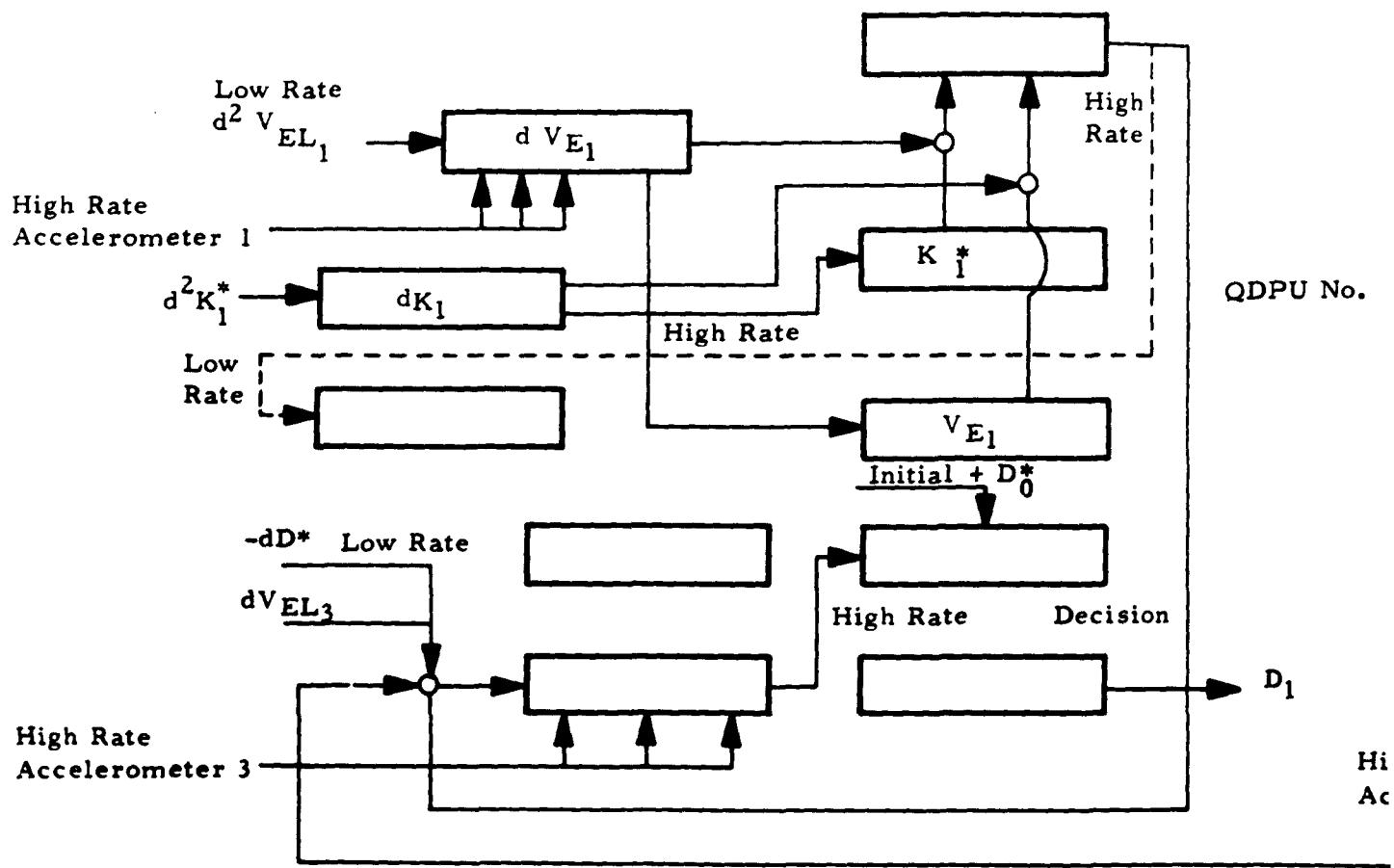
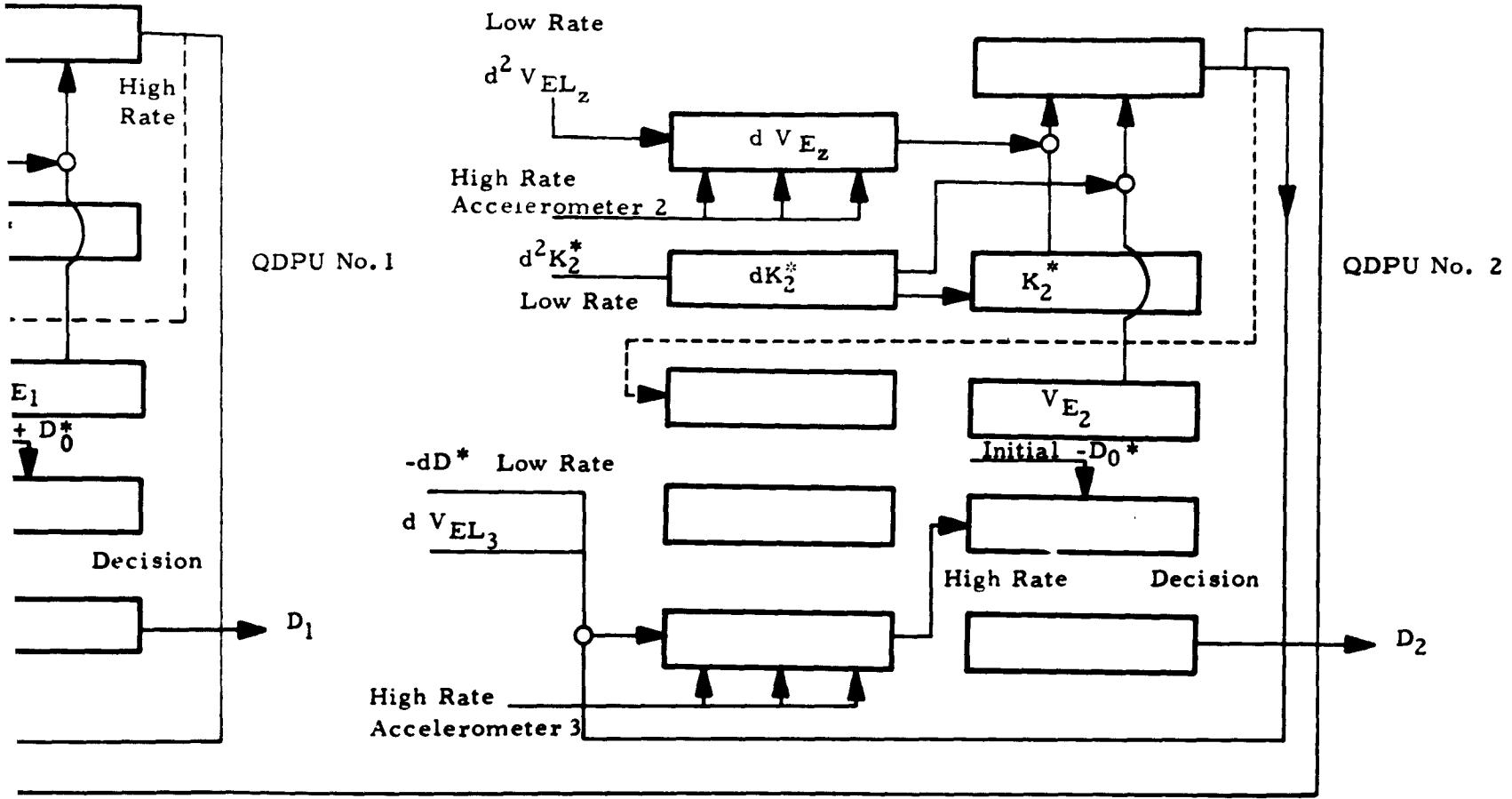


Figure 11-7. QDPU Program for Thru-Rate Computations of Input Computation Routines

1



- ## 7. QDPU Program for Thrust Cut-off by Multi-Iteration Rate Computations of Input Processing and Internal Computation Routines



- NOTE:** (1) Multi-increment dK_1 , dV_{E_L} quantities (provided at low rate) are used at high rate to obtain high rate accuracy.
- (2) Formal integrator equivalent, neglecting high rate computations, with dV_{E_L} , dK quantities is $2QDPU = 6$ DDA Int. An ordinary DDA could not reduce granularity of dV_{E_L} , dK which have large program calculations without lowering iteration rate an unacceptable amount.

The QDD³A performance, in the thrust cut-off function, is estimated for a number of input processing iteration rates for which additional input processing functions may be executed (at the same rate) which have program lengths expressed in DDA integrator count which total less than or equal to tabulated values on the basis of internal computation at 100 iter/sec with >200 DDA integrator program count (assuming 1.4×10^3 words/sec):

Input Processing Iteration Rate	Thrust Cut-off RMS Error	Residual Input Processing Programming Space
4500 iter/sec	0.03 ft/sec	0
1400	0.15	12 to 17 DDA Integ
700	0.35	35 to 44
400	0.60	60 to 80

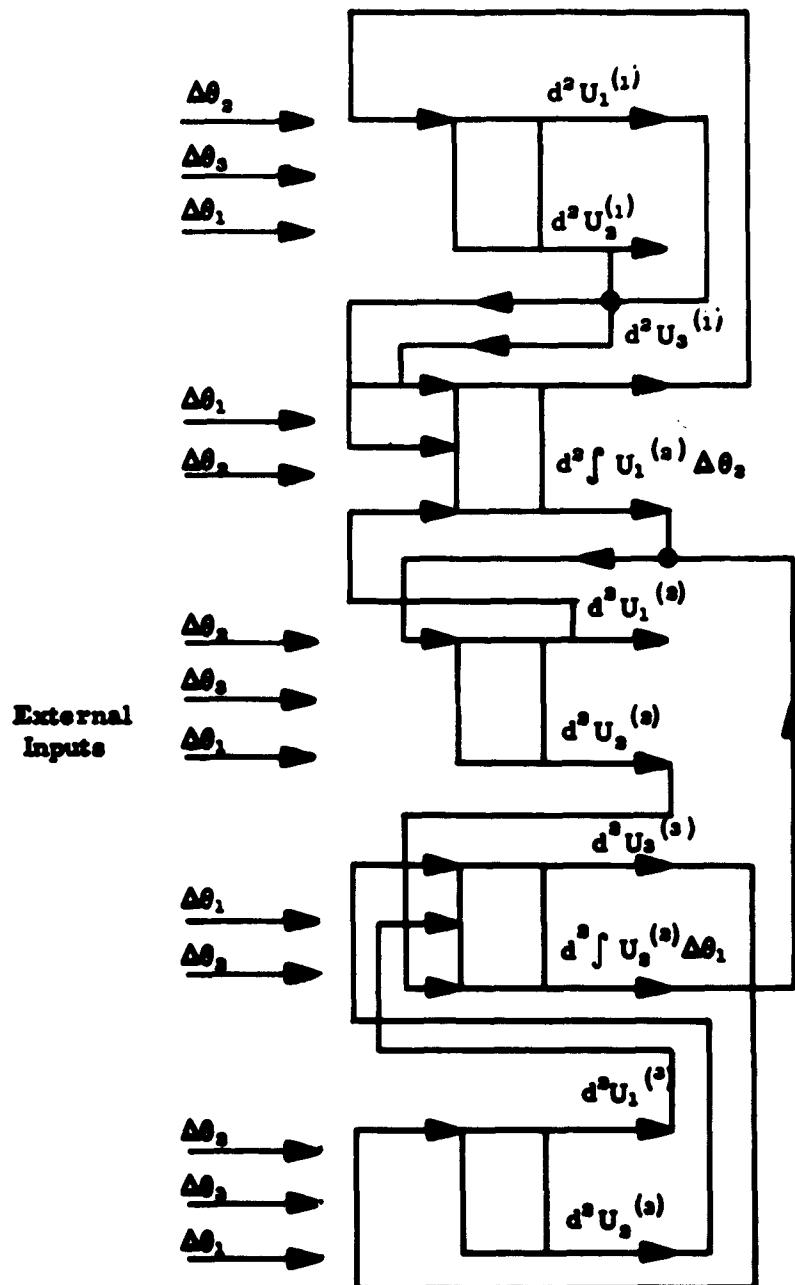
15 g Thrust Acceleration assumed (for the 10g case look at rows with iteration rate 1.5 times higher for performance)

The estimate of residual input processing program space assumes 1 QDPU \cong 4 DDA Integ. in internal computation and uses the formula (Word Rate) = $(IR_{Imp\ Proc}) (N_Q Residual Imp Proc^2) + (IR_{Int\ Comp}) (N_Q Int\ Comp)$ where N_Q is a number of QDPU.

Assuming that the tolerable rms error in thrust cut-off is 0.4 ft/sec, it is seen that, an extensive amount of additional input processing is available at >700 ft/sec for a 14,000 word rate.

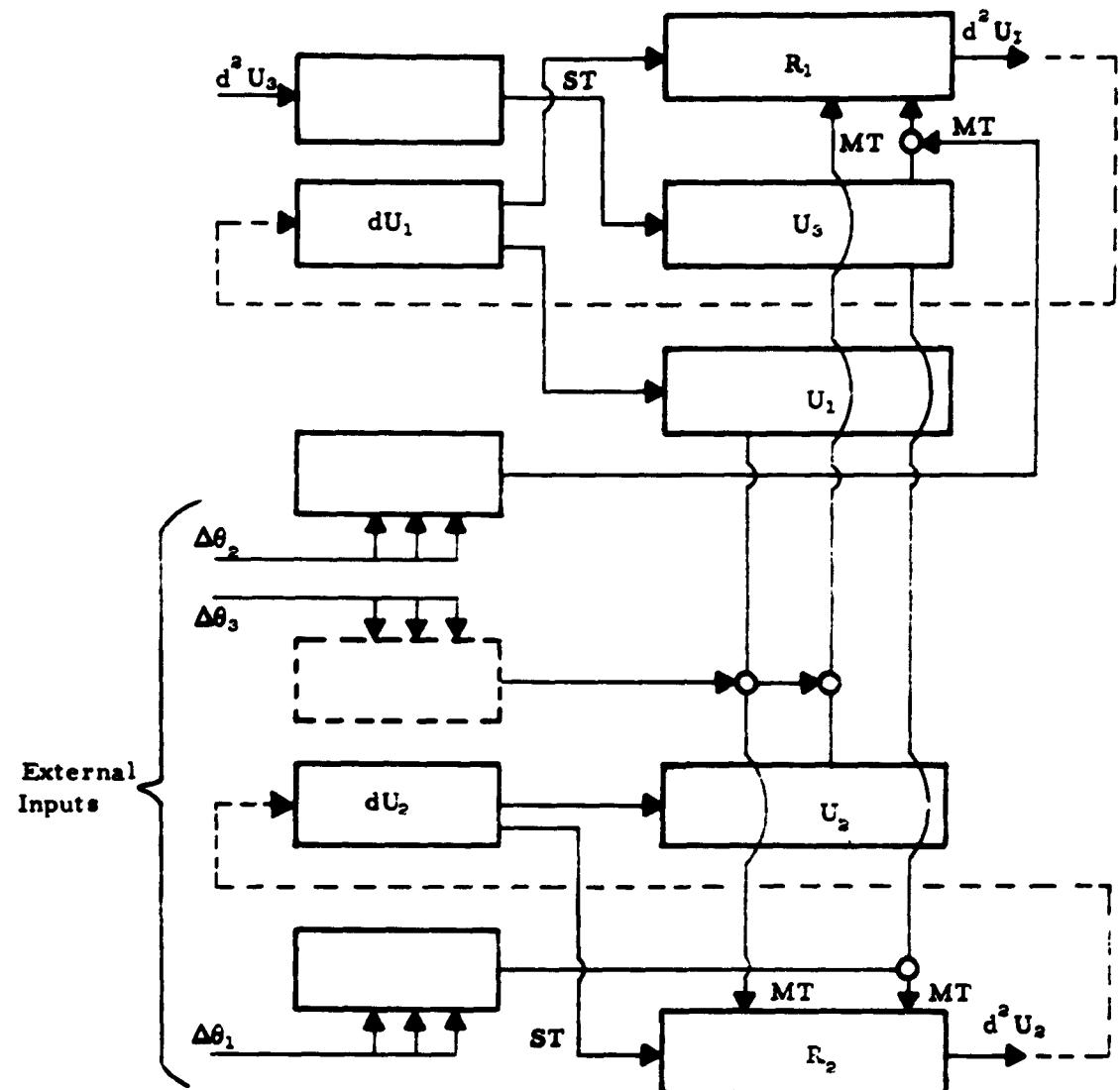
- C. Strap-Down Computations in the High Rate Input Processing Loop of the QDD²A - Input processing for strap-down computations is obtainable at very high rate in the QDD²A using multi-iteration rate time-sharing with internal computation. The case considered is that for pulse stream transducers for the rate gyros (maximum pulse rate 10⁴/sec) and state of the art bit rates for the digital computer. Figures 11-8, 11-9 and 11-10 show that strap-down computations for inertial reference (inertial velocities are obtainable with required precision in internal computations using the strap-down computations) can be programmed for multi-increment computation (single precision*) in 5 QDPU. At 1270 ft/sec the input angular increment is only 3 bits multi-increment hence single precision effected by 3 bit or 4 bit multi-transfer is entirely adequate for state of the art pulse stream transducers. Thrust cutoff calculations requiring 2 QDPU in the same input processing loop of 11 word time cycles add to input processing requirements which

*Single precision is 3 bit increment computation in proposed QDD A.



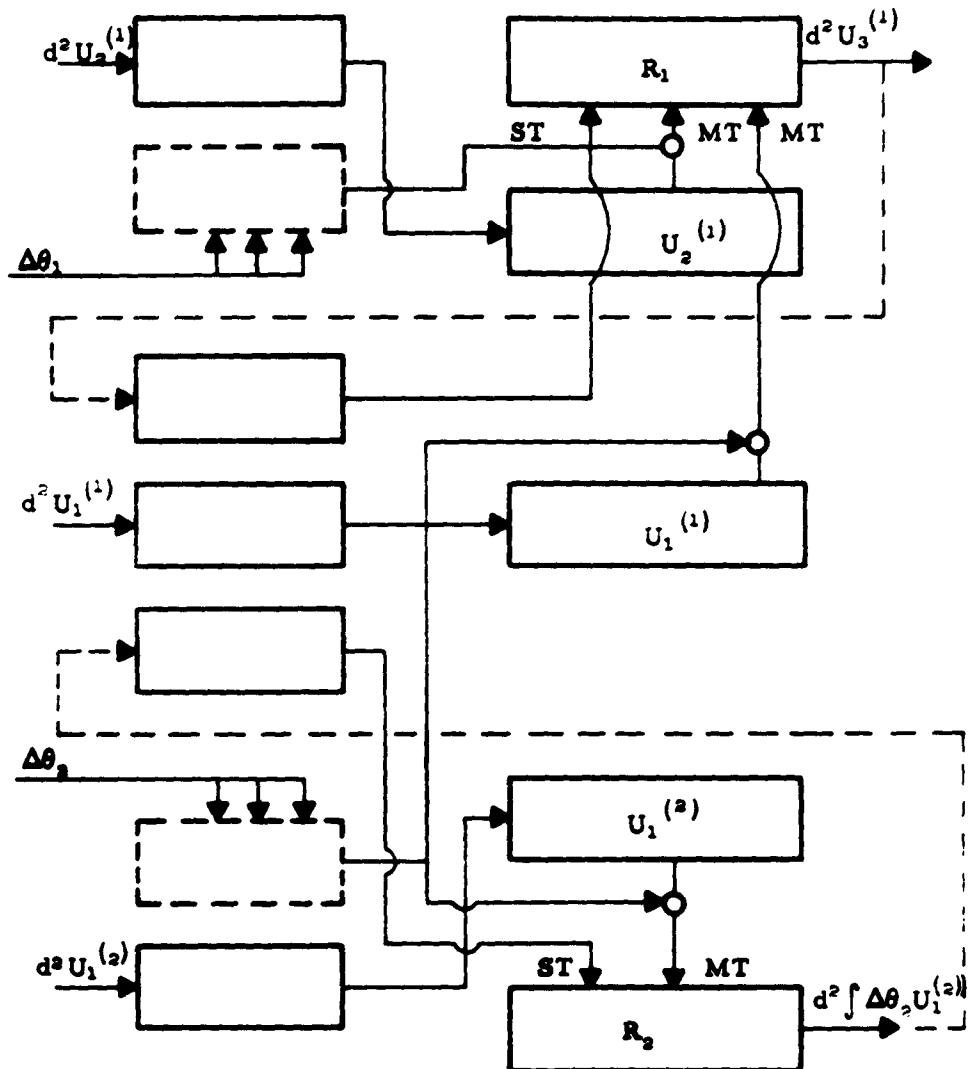
Strap-Down Computations and Thrust Cutoff
Computations Executed at 1270 it/sec (0.32 meg bits/sec hardware).
Internal Computations at 90 it/sec. Total integrator equivalent of
program, 256 DDA Integrators.

Figure 11-8. Strap-Down Computation QDPU Program (Craft Orientation in Inertial Space)



External inputs have their own registers. The QDPU program code for $I_1 = I_2 = 1$ (contradictory since it calls for the same quantity in two y registers, hence defines modal action).

Figure 11-9. Strap-Down Computation QDPU Program Type for 3 of 5 QDPU



External Inputs have their own registers. The Δx of MT_1 , MT_2 , MT_3 is programmed for external input by the selection $I_1 = I_2 = 1$ and Δx selection code excluding 6 registers ($I_1 = I_2 = 1$ is contradictory since it calls for two y registers holding the same quantity, hence defines modal action).

Figure 11-10. Strap-Down Computation QDPU Program Type for 2 of 5 QDPU

total 7 QDPU. In time-sharing with internal computation this yields 1270 it/sec input processing, 90 it/sec internal computation assuming, state of the art, 320,000 bit hardware. Programmed internal operation of QDPU is of two types for strap-down computations as indicated in the diagrams. External inputs are drawn from the preprocessing loop which collects pulses as a buffer, emitting accumulated angular increments with fixed format to the QDDA where used as associated Δx of each multi-transfer unit when the QDPU program instructs no selection of δ_1 , δ_2 , δ_3 register for Δx .

D. Evaluation of QDD²A For Computations of Energy Management During Re-entry

1. Computation Structure and Comparison of GP and QDDA Performance - The re-entry computation program (assuming the unified control approach as formulated by Daniel Dommasch) is very large (of the order of 250 DDA integrators). Re-entry is the most critical phase of the entire mission since the re-entry corridor is quite narrow due to structural limitations and maximum allowable heat input to the vehicle. The pitch acceleration command computation, essential to temperature control, involves decision selection of functions involving division and square root which are shown to be very efficiently generated in the QDDA (analyzed in Chapter XII). External inputs important in the energy management computations, are pitch and roll

angles, wall temperature and the time rate, and air density and altitude. These quantities change at relatively high rates during re-entry and present an insurmountable computation problem to a conventional DDA in generating the extensive vector transformations, spherical triangle solutions, energy and aerodynamic computations of the program, and the final command signal calculations. Recent efforts in aerospace guidance and control have been predicated on allocation of the bulk of these calculations to the GP section. The copious supply of sin, cos, arc sin, arc and arc tan calculations in the re-entry program, each one of which requires 0.4 milliseconds in advanced GP with fast multiplier and clock rate pressing the state of the art, are a contributing factor to the low iteration rate of the GP program (10 iter/sec). The QDD³A can increment these functions in the indicated word times by executing the indicated operations:

$$\begin{aligned} d \sin \theta &= \cos \theta d\theta \\ d \cos \theta &= -\sin \theta d\theta \end{aligned} \quad \left. \begin{array}{l} \text{Sinusoid} \\ \text{(Double Precision for Inputs)} \end{array} \right\} 1 \text{WT} \quad (\text{XI-54})$$

$$d\theta = \frac{dx}{\sqrt{1-x^2}} \quad \left. \begin{array}{l} \text{Arc sin } x \\ \text{(Single Precision non-input)} \end{array} \right\} 1 \text{WT} \quad (\text{XI-55})$$

$$d\sqrt{1-x^2} = \frac{-xdx}{\sqrt{1-x^2}} \quad \left. \begin{array}{l} \text{Arc cos } x \\ \text{(Single Precision non-input)} \end{array} \right\} 1 \text{WT} \quad (\text{XI-56})$$

$$d\sqrt{1-x^2} = \frac{-xdx}{\sqrt{1-x^2}} \quad \left. \begin{array}{l} \text{Arc cos } x \\ \text{(Single Precision non-input)} \end{array} \right\} 1 \text{WT} \quad (\text{XI-57})$$

$$d\theta = \frac{-dx}{\sqrt{1-x^2}} \quad \left. \begin{array}{l} \text{Arc Cos } x \\ \text{(Single Precision non-input)} \end{array} \right\} 1 \text{WT} \quad (\text{XI-59})$$

$$d\sqrt{1-x^2} = \frac{-xdx}{\sqrt{1-x^2}}$$

$$d\theta = \frac{dx}{1+x^2} \quad (XI-60)$$

} Arc tan x
(Single Precision non-input) 1 WT

$$d(1+x^2) = 2xdx \quad (XI-61)$$

Without pressing, the state of the art, a word rate of 20,000 words/sec, the QDDA can update any of these quantities in 0.05 milliseconds which implies a speed advantage of a factor of eight over the advanced GP. For operations involving products the speed advantage can be deduced as follows. The example GP has an operation rate (for multiplication) of about 60 μ sec. The QDDA on the average performs computations involving products with the capability of 4DDA integrators per word time executing 2 product increments in parallel. Thus for a 0.05 millisecond word time the average product time 25 μ sec implying a 2.4 factor of rate advantage for the QDDA for products. A conventional DDA can add variables as integrands without cost in time but not independant variables which require quantization. The QDDA does both without cost in computation time. The comparison GP requires an operation time for add or subtract hence it is just as slow as for multiplication. Assuming half the operations in re-entry are add or subtract the net speed advantage for add, subtract, multiplication is a factor of 4.8. Including the trigonometric performance the net speed advantage in computations on continuous variables is a factor of 5 to 8 or say 6.5. This estimate holds only for

variables which are well defined throughout the desired phase of operation. The problem, arising from singularity and other discontinuity of variables is a most prominent DDA programming and design problem. The approach of allocating the bulk of real time computations to an incremental computer offers the possibility of a mechanization with simplified GP or special computing unit of reduced complexity given the primary task of conditional whole word communication and discontinuity handling functions. Since in critical cases a singularity is the result of a coordinate system, the occurrence of singularities is usually at easily located points. Usually only one source of singularity occurs within a given comparatively long time interval.* Therefore, a simplified GP or special computing unit of modest rate capability could devote, essentially, full time to handling a single singularity effect in a program branch. Certain DDA computations with discontinuities can be handled using decision modes, specifically, including those in which cognizance of the valuelessness of a singularity variable at the time of singularity is taken into account by ignoring computed values; permanent error is avoided by special programming. A clearcut example of this capability is given by the QDD⁸A doppler damping program presented in section XII. Decision modes are essentially free in processing time in the QDD⁸A since full

*Doppler damping program analysis shows how the QDD⁸A can handle an important problem in which this condition is not required.

arithmetic capability (none being required in decision modes) is utilized in parallel with decision operation. While the scope of this study program has not permitted full evaluation of all specific singularity problems which arise in aerospace applications, it is believed that by careful programming, utilizing the principles exemplified in the doppler damping QDDA program, it may be possible to obviate use of the GP for computations involving isolated singularities. The extent of additional programming to accomplish this result, if appreciable, would appear to be <20 percent at most. A minor mechanization elaboration for step changes as occurs in longitude can be implemented by decision command for sign reversal of a y register variable e. g. from +w to -w longitude. The major speed advantage of the QDD²A over GP is in this case retained in the proposed system of reduced hardware complexity.

E. QDD²A Program For Total Pitch Acceleration Command In Re-entry - The structural limitations and maximum allowable heat input to the vehicle are critically determined by the total pitch acceleration commanded by the digital computer during re-entry. A unified control approach developed to guide the missile to target point within these constraints involves a total pitch acceleration command computation that uses decision selection between two closely related command alternatives. These command alternatives computations may be expressed in the form

$$\ddot{\theta}_{\pm} = \pm \frac{y}{\sqrt{x/\lambda - x}} \quad \text{where } 0 < x < \lambda = \text{const (XI-62)}$$

where the selection of sign is according to the test

$$F(t) < x$$

It will be shown that the QDPU is highly efficient in executing the basic control function as a result of capability of parallel divisions with common divisor in the single QDPU.

Introducing the notation $\sqrt{x/\lambda} = r$ obtain,

$$\ddot{\theta}_{\pm} = \frac{y}{\pm r - x} = \frac{y (\pm r - x)}{(\pm r - x)(\pm r - x)} = \frac{y (\pm r - x)}{x(x - 1/\lambda)} \quad (\text{XI-63})$$

The differential of $\ddot{\theta}_{\pm}$ may be shown to have the form

$$\ddot{d\theta}_{\pm} = \frac{-\dot{\theta}_{\pm} dx - du_{\pm}}{x - 1/\lambda} \quad (\text{XI-64})$$

where $dU = dy \pm d(W)$

$$dW = d(yR)$$

$$dR = 1/\sqrt{\lambda x}$$

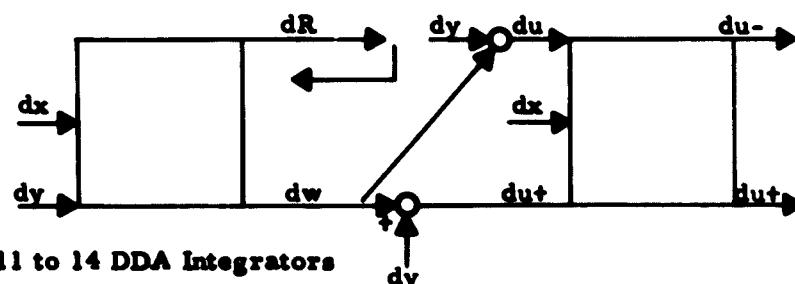
The calculation form for $d\ddot{\theta}_+$ and $d\ddot{\theta}_-$ is seen to be obtained in the common divisor form. Further, because of the natural feedback of output, the only programmed inputs are x and u for each calculation, which are the same. A single QDPU can therefore generate $\ddot{\theta}_+$ and $\ddot{\theta}_-$ from x and u using the four

transfer capability in single precision mode (3 bit multi-increment in the proposed QDDA). The computation of W and R is given by

$$dR = -\frac{Rdx}{2x} \quad (\text{XI-65})$$

$$dW = ydR + Rdy \quad (\text{XI-66})$$

Both of these calculations are executed together by one QDPU. Selection of θ_+ or θ_- is provided by the ordinary decision command which, in the QDPU, is executed in parallel with a double precision calculation because it is an essentially free operation. The incrementation of θ_+ and θ_- given x, u in one QDPU requires one word time, whereas 6 to 8 word times are required by a conventional serial DDA. Even if executed in one word time in parallel conventional single increment DDA, the result would be orders of magnitude less accurate because of the relatively high rates of change of x. The generation of R, W given x, y by one QDPU in one word time has a 5 or 6 DDA integrator equivalent. Figure 11-11 shows a schematic of the QDPU program.



2QDPU = 11 to 14 DDA Integrators

Figure 11-11. Schematic of the QDPU Program for Pitch Command During Re-entry

F. Challenging Airborne Digital Computer Routines and Possible Aerospace Military Function Computations

- 1. Introduction - A number of airborne computation tasks are not successfully handled by conventional DDA including:**
 - a. Doppler damping in airborne inertial navigation.** This computation which presents problems that may prove important in aerospace applications, is analyzed in detail in the next section and programmed for the QDD²A as a subroutine in a full aerospace mission computer. A computer specially designed for airborne navigation alone could be designed to meet state of the art accuracy requirements by elaborating conventional DDA with the developed digital Stieltjes integration algorithm and 2 bit increment computation.
 - b. Toss Bombing and Fire Control -** These computations require high rate variable computations with modest to intermediate accuracy requirements over short periods. Multi-increment computers surpass the conventional DDA in rate handling capability by variable (single) increment computation, however, where more than modest accuracy over short term is required, the multi-increment QDD²A is clearly called for. The toss bombing program presented in Chapter VI is directly applicable to the multi-increment QDD²A (by communicating second rather than first

differences to the modified QDPUs). The programming of double precision mode in input processing associated with earlier stages of toss bombing computations would offer an additional level of accuracy improvement. Fire control computations involve computation requirements similar to toss bombing bombing but, in certain cases, could be much more demanding from the standpoint of accuracy, therefore could require the QDD²A.

- c. Digital Autopilot and Replacement of Stable Platform Analogue Computers - Mechanization savings are possible by use of a digital computer capable of handling such functions as autopilot and stable platform leveling.

The possible aerospace military computation functions are surmised as similar in computation structure and computation requirement types to the challenging airborne digital computation tasks enumerated above. The requirements are expected to be more demanding.

Most of the computation problems of the class of computation routines discussed in this section are similar to those in the doppler damping problem analyzed in the next section.

G. Doppler Damping Computations For Conventional Navigation (Multi-Increment QDDA)

1. Introduction - The conventional DDA is at best marginal* for that essential function in long term navigation (> 2 hr): doppler damping. Reasons for low capability in doppler damping in a DDA is (1) Damping is executed using doppler velocities measured in rapidly changing craft coordinates which must be transformed to inertial coordinates (2) Operation is subject to periods of valueless information (for damping) which can result in large errors if damping is used when craft orientation brings the resultant doppler beams to vertical orientation**. Several conventional computation approaches using DDA-GP combinations have been found lacking. They involve either:
 - (1) The approximation that craft orientation is on the average approximately horizontal and can be assumed so continuously (leading to poor results in flight test), or
 - (2) The supervision of the DDA by the general purpose computer in error sensitive portions of the routine. As the GP has a large program generally it turns out that the iteration rate of the GP may be so low that the GP supervised variables change excessively between iterations making the supervision inadequate.

* Marginal in low accuracy navigation, inadequate in high accuracy navigation.

**Despite the fact that a conventional DDA has equivalent decision modes.

The approach of GP supervising several times per GP iteration reduces this inadequacy, but at a corresponding price in programming length in an already cramped system, or by further reducing GP iteration rate. In the chapter on computation capability the conventional DDA is shown to be lacking even when damping accuracy is assumed to be substantially sacrificed.

The QDD²A will be shown ideally suited to doppler damping in conventional navigation without GP supervision by virtue of QDD²A features of:

- (1) Multi-increment computation
- (2) Relatively high iteration rate
- (3) Decision modes enabling damping turn off and pseudo variable computation (described below)
- (4) Double precision mode for error sensitive computations e. g. sinusoid.

The quantitative performance of the QDDA is given along with conventional DDA in the referred to computation capability analysis. The equations for doppler damping which later will be put in form for QDDA computation involve use of altimeter information to obtain vertical velocity information.

2. Raw Equations for Doppler-Altimeter Deduced Inertial Velocity - Two doppler beams in rigid craft coordinates

$$\bar{A}_{s, p} = A\bar{x} \pm C\bar{y} + B\bar{z}$$

(XI-67)

where $A = -\sin x$, $B = \sin x \cos v$, $C = \cos x \cos v$
yield from the doppler radar transducer, the starboard
and port components

$$S = \frac{\lambda}{4} \frac{(P_s + P_p)}{\Delta t}, \quad P = \frac{\lambda}{4} \frac{(P_s - P_p)}{\Delta t} \quad (\text{XI-68})$$

from pulse rates P_s , P_p which are related to velocity
components

$$\frac{P_s}{\Delta t} \frac{\lambda}{2} = AV_x^- + CV_y^- + BV_z^- \quad (\text{XI-69})$$

$$\frac{P_p}{\Delta t} \frac{\lambda}{2} = AV_x^- - CV_y^- + BV_z^- \quad (\text{XI-70})$$

hence

$$S = AV_x^- + BV_z^-, \quad P = CV_y^- \quad (\text{XI-71})$$

Craft to earth velocity vector transformation is

$$V_x = V_x^- \cos \theta + V_y^- \sin \theta \sin \phi + V_z^- \sin \theta \cos \phi \quad (\text{XI-72})$$

$$V_y = 0 + V_y^- \cos \theta - V_z^- \sin \theta \quad (\text{XI-73})$$

$$V_z = -V_x^- \sin \theta + V_y^- \cos \theta \sin \phi + V_z^- \cos \theta \cos \phi \quad (\text{XI-74})$$

where $V_z = \frac{dh}{dt}$, h barometric altitude. Doppler-
altimeter deduced velocities in computer inertial
coordinate orientation,

$$V_{X_D} = V_x \cos x - V_y \sin x \quad (\text{XI-75})$$

$$V_{Y_D} = V_y \cos x + V_x \sin x \quad (\text{XI-76})$$

3. Computation Problems and Special Computation Methods

An important feature of the doppler damping equations is that, for certain craft orientations (primarily pitch angle up so that radar beam is vertical) the equations do not yield a solution for horizontal velocity. The successful use of a DDA in performing these computations clearly requires that the DDA be capable of decision functions as well as high rate handling capability. The specific decision functions required are:

- (1) Damping turnoff when craft orientation renders doppler-altimeter deduced velocity unreliable.
- (2) Pseudo variable computation in place of a variable with unacceptable analytical properties for DDA computation: i. e., the replacing of a variable with a decision modified variable which is equal to desired variable when the variable is used (during damping) and when the variable is not used is a well behaved variable for accurate DDA computation. Clearly, the pseudo variable computation approach is a fundamental one in broadening the scope of DDA application.

The stated implicit form of the damping equations could in principle be solved only by a DDA with servo mode. The generally low accuracy performance of servo computation is further degraded to unfeasibility for this application because of the peculiar analytical character of the calculations for certain craft orientations. The orientation sensitivity is revealed by the explicit form of the equations stated:

$$V_y = P/C, V_{Z_B} = -\frac{dh}{dt} \quad (\text{XI-77})$$

$$V_x = \frac{1}{\mu} \left[-\lambda V_{Z_B} + V_y \sin \theta + \frac{S}{B} \cos \theta \right] \quad (\text{XI-78})$$

$$V_y = \frac{1}{\mu} \left[-v V_{Z_B} + V_y \sin \theta - \frac{S}{B} \sin \theta \sin \theta \right] \quad (\text{XI-79})$$

where

$$\mu = \sin \theta + \frac{A}{B} \cos \theta \cos \theta$$

$$\lambda = \cos \theta - \frac{A}{B} \sin \theta \cos \theta$$

$$v = \frac{A}{B} \sin \theta$$

$$\xi = \frac{A}{B} \cos \theta$$

$$c = \sin \theta \cos \theta + \frac{A}{B} \cos \theta$$

Note that $\mu = 0$ for $\cos \theta \approx 1$ when $\theta = \theta_c$ is such that $\tan \theta_c \approx -\frac{A}{B}$,

a situation which can occur with relatively high frequency during craft maneuver or bad weather. When the critical pitch angle θ_c is approximately taken by the craft two error effects degrade computation accuracy:

- (1) Scaling of a reciprocal calculation if used is such that granularity is unacceptable. Looseness of a servo loop, if used in implicit calculation, degrades accuracy.
- (2) The factors $1/\mu$ of variables of each of three independent information sources (two doppler velocities and altimeter), each subject to independent errors, cause unacceptable noise amplification for $\mu \approx 0$.

In general, for a computer, a decision mode must cut off damping in consequence of error effect (2). In order to prevent rate limiting or servo loop instability, which can permanently nullify the computation accuracy after a transition through $\mu = 0$, it is necessary to either reset DDA computed variables (expensive and usually inadequate), or compute in pseudo variables (defined above). The QDD²A may always compute in terms of amenable variables provided $1/\mu$ may be available when damping is on. The other variables require high rate handling capability which the QDD²A is designed to possess. Generation of $1/\mu$ when damping is on does not require that $1/\mu$ be generated when damping is off. Because the craft orientation pattern is not a priori known certainly μ must be available with good accuracy at all times. These facts together with the requirement of overcoming error effect (1) suggest the computation of the variable $1/\mu^*$ where

$$\begin{array}{ll} 1/\mu^* = 1/\mu & \mu \geq \mu_o \\ & \mu \geq \mu_o \\ = 1/\mu & \mu < \mu_o \end{array}$$

choosing damping turn off when $\mu < \mu_o$, where μ_o is an appropriately chosen constant. It is seen that $1/\mu^* = 1/\mu$ when damping is on hence no approximation is made during periods of useful damping information. Appropriate μ_o selection also ensures that no scaling problem is presented in reciprocal computation. This type of computation used in $1/\mu^*$ computation might be called function limiting. The QDDA has a function limiting mode which is highly efficient in equivalent integrator performance because several other operations may be performed in parallel. The hardware cost of the mode is minor because the other parallel operations are achieved in the normal QDD²A program.

4. QDDA Calculations Involving Decision Modes for the Doppler Damping Case - The QDDA computations of the primary feedbacks dD_x, y for doppler-altimeter damping should include the following:

$$dD_x = K_1 dC_x^* \quad (\text{XI-80})$$

$$dD_y = K_1 dC_y^* \quad (\text{XI-81})$$

where

$$dC_x^*, y = \begin{cases} dC_x, y & \text{if } u \geq u_0 \\ 0 & \text{if } u < u_0 \end{cases}, d\alpha^* = \begin{cases} da & \text{if } u \geq u_0 \\ 0 & \text{if } u < u_0 \end{cases}$$

$$dC_x = \frac{(a V_x dt) - \alpha (V_x I dt)}{a^*} \quad (\text{XI-82})$$

$$dC_y = \frac{(a V_y dt) - \alpha (V_y I dt)}{a^*} \quad (\text{XI-83})$$

the quantities $V_x I dt$, $V_y I dt$, being the inertial velocity increments computed in the inertial navigation routine, and αV_x , αV_y being the doppler-altimeter deduced quantities of the damping routine formed by quantizing component terms separately computed. Were computation precise, then when $a^* = a$, $dC_x, y = \frac{(a V_x, y dt)}{a^*} - V_x, y I dt$. The chosen computation procedure is preferred rather than computation

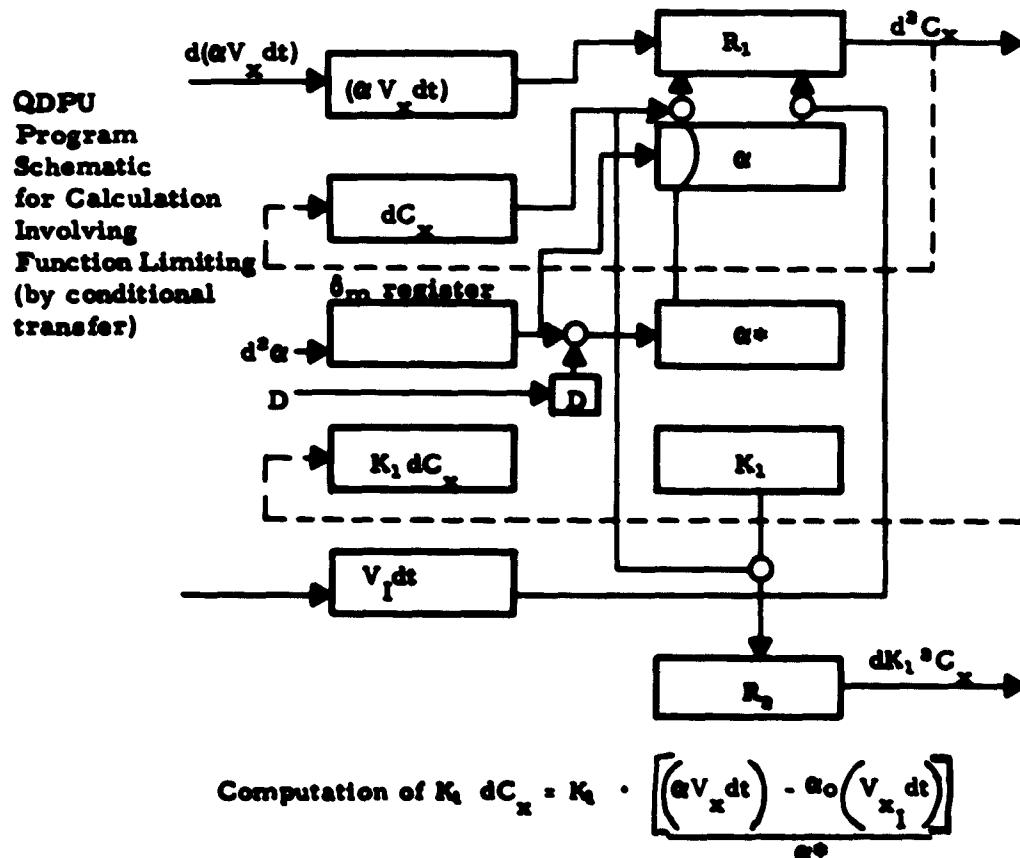
$$dC_x, y = \frac{(a V_x, y dt)}{a^*} - V_x, y I dt \quad (\text{XI-84})$$

which requires less programming. This occurs because any long term errors which develop in division, in the chosen procedure, have the unimportant effect of changing the damping constants slightly rather than destroying the accuracy of doppler deduced error velocity as in the simpler computation.

The QDPU has the capability of utilizing one decision command signal (per word time). A marked input to the δ_m register identifies the decision command signal D (where D = 1, 0) which then, according to programmed decision mode of the QDPU, modifies transfer action of δ_m register contents after the unmarked inputs to δ_m have been used to update the δ_m register. This readily mechanized design feature accords especial efficiency to the QDPU in the decision modes since otherwise normal programmable inputs distributed to the three δ registers presents the maximal variables for operation, and otherwise normal programmable transfer action retains full operations versatility. The QDPU program schematics for operations involving function limiting and signal cut-off for the doppler damping application, are presented in Figures 11-12 and 11-13 on the following pages. In this example of function limiting, 2QDPU perform* the program routine of 10DDA integrators (duplicated operations in K_1/a^* calculation reducing DDA requirements from 14 to 10). In the case of signal cutoff operation, exemplified

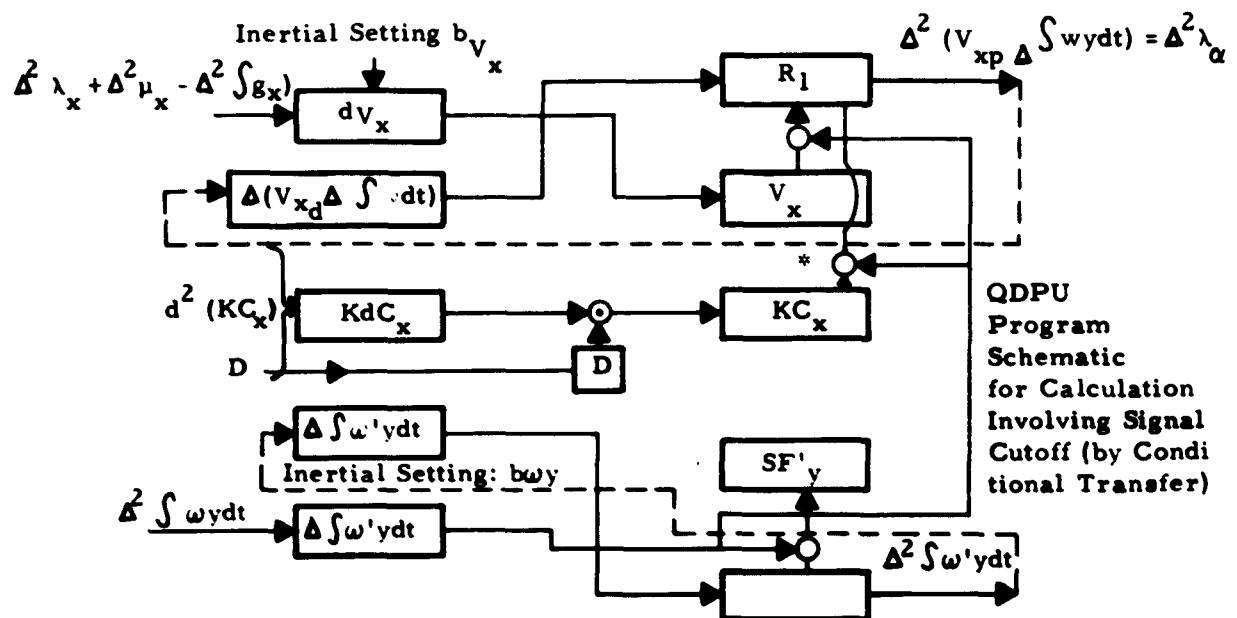
*In terms of hardware processing rate (actually the greater precision of QDD'A by multi-increment computation and algorithm amounts to an additional speed advantage).

in the second QDPU schematic, the efficiency would probably be reduced somewhat from the latter performance in certain other examples than doppler inertial damping. The capability (and requirement in this case) of whole word drift correction called for external input, external output quantities, leading to efficiency of 1QDPU for 5 DDA integrators.



1 QDDU = 7 DDA Integrators

Figure 11-12. Doppler Damping Without GP Supervision (Multi-Increment QDPU Using Decision Mode)



**Figure 11-13. Doppler Damping Without GP Supervision
(Multi-Increment QDPU using Decision Mode)**

Computations: Doppler Damped System Velocity with Automatic Damping Damping Turnoff on Decision; and Drift Rate Corrected and Scaled Gyro Rate Signal Calculation.

NOTE: 2DDA Integrators equivalent accorded to whole word drift rate correction of input and output of the computer. Since in a conventional DDA, an R register initial bias of an integrator generating the variable du merely makes the quantity u biased it is seen that if u had drift error $-b_u t$ that an extra DDA integrator is necessary to generate the correction $+b_u t$. In the QDDA, the correction is essentially free.

In the decision mode, the first input to δ_m register picked up is used as a decision variable to modify transfer of δ_m contents formed by updating with the remaining inputs to δ_m . The decision command variable is generated in the parallel channel of input processing calculations since no multi-transfer operations are required for decision nor available in fully efficient input processing.

The $K_1 dC_y$ computation may be regarded as having only the performance 1 QDPU \equiv 3 DDA Integrators since in a DDA the quantity $d(K_1/a^*)$ would be available in $K_1 dC_x$ calculation. The subroutine is evaluated 2 QDPU \equiv 10 DDA Integrator.

5. Single Precision and Double Precision Calculation Allocation - One of the most powerful features of the QDD²A design is that it enables, according to the particular computation requirements of a calculation routine, the allocation of appropriate computation capacity to meet those requirements. Analogous to the general purpose computer, which is capable of double precision programming, the proposed incremental computer may use brute force where necessary and not otherwise. Because the QDD²A has been designed on a more fundamental logical level than the conventional GP it can achieve double precision by arithmetic unit transformations rather than by data flow programming. Thus, double precision is obtained with only a reduction factor of 2 in instantaneous processing rate, whereas in a GP the

the reduction is 4 to 7*. In the particular problem of doppler damping, analysis indicates that double precision (input processing) is not necessary throughout the calculation because 0.2 percent accuracy is quite adequate. Special subroutines, however, require double precision which are error growth sensitive and which enter in the computations in such manner that damping does not attenuate error buildup. The sinusoid calculations on input angles are the case in point and have been allocated double precision.

6. Doppler Damping Program for QDD²A Computation - The doppler-altimeter deduced inertial velocity calculations and a portion of the blended calculations for feedback and inertial operations are summarized below in differential form. QDPU allocation is indicated and evaluated relative to the conventional DDA (in cases where the relative evaluation is deducible only by sets of QDPU, corrections are interspersed to give the correct total evaluation).

*The design approach developed in this study could be applied to the design of a GP with adaptable precision and maximal speed performance.

$$d(V_x d \int \omega_y dt) = d\lambda_z = [V_x + b V_x^t + KCD \underset{\substack{\text{Decision} \\ \text{Cutoff}}}{\underset{\substack{\text{Inertial Accel Bias Corrected}}}{}}] d \int \omega_y dt \quad IQDPU = 5DDA \quad (XI-85)$$

$$d \int \omega_y' dt = SF_y' d \int \omega_y dt + b \omega_y dt \quad (XI-86)$$

External Output
(Gyro Torque)

Analogous for $x \rightarrow y, y \rightarrow x$

$IQDPU = 5DDA$
Integrators

$$dK_1 C_K = K_1 \frac{[(\alpha V_x dt) - \alpha \omega (V_x dt)]}{\alpha}$$

$IQDPU = 7DDA \quad (XI-87)$
Integrators

Analogous for $x \rightarrow y$

$IQDPU = 3DDA$
Integrators

$$\text{where } \left[\alpha (V_x - V_{x_I})^{dt} = du_x + d\xi_x + v_x - [\alpha \omega V_{x_I} dt] \right]$$

(XI-88)

and V_{x_I} is inertial velocity

$$dB_{CC} = \frac{\cos \phi d \cos \theta + \cos \theta d \cos \phi}{\left(\frac{A}{B}\right)^{-1}} \quad (XI-89)$$

IQDPU ≈ 5DDA
Integrators

$$d\epsilon_x = -\cos \phi \left[\frac{(P_s + P_p) dt}{4\Delta t B/\lambda} \right] \quad (XI-90)$$

$$dB_{CS} = \frac{\cos \phi d \cos \theta + \sin \theta d \cos \phi}{\left(\frac{A}{B}\right)^{-1}} \quad (XI-91)$$

IQDPU ≈ 5DDA
Integrators

$$d\epsilon_y = \sin \phi \left[\frac{(P_s + P_p) dt}{4\Delta t B/\lambda} \right] \quad (XI-92)$$

$$\begin{aligned} d(V_x \cos x) &= dV_x \cos x + V_x d \cos x \\ d(V_y \cos y) &= dV_y \cos y + V_y d \cos y \end{aligned} \quad \left. \begin{array}{l} \text{IQDPU ≈ 4DDA Integrators} \\ \text{IQDPU ≈ 4DDA Integrators} \end{array} \right\} \quad (XI-93)$$

$$\begin{aligned} d(V_x \sin x) &= dV_x \cdot \sin x + V_x \cdot d \sin x \\ d(V_y \sin y) &= dV_y \cdot \sin y + V_y \cdot d \sin y \end{aligned} \quad \left. \begin{array}{l} \text{IQDPU ≈ 4DDA Integrators} \\ \text{IQDPU ≈ 4DDA Integrators} \end{array} \right\} \quad (XI-94)$$

$$dV_x = \frac{\left[-\cos \theta - B_{CS} \right] d \int \frac{A}{B} s_{V_z_B} \left(V_z_B dt \right)}{\frac{A}{B}} \quad \left. \begin{array}{l} \text{IWT ≈ 3DDA Integ} \\ \text{IWT ≈ 3DDA Integ} \end{array} \right\} \quad (XI-95)$$

$$d \int \frac{(P_s + P_p) dt}{4\Delta t B/\lambda} = \frac{(P_s + P_p) dt}{4\Delta t B/\lambda}$$

$$\left. \begin{aligned} \frac{du}{y} &= \sin \phi \left(d\eta_y + d\xi_y \right) + \sin \phi d \int \frac{A}{B} S_{V_{z_B}} (V_{z_B} dt) - \frac{A}{B} d\eta_y \\ d \int \frac{A}{B} S_{V_{z_B}} (V_{z_B} dt) &= \frac{A}{B} S_{V_{z_B}} \cdot (V_{z_B} dt) \end{aligned} \right\} \quad \begin{aligned} 1 \text{ WT} &\equiv 4 \text{DDA} \\ (\text{XI-96}) \end{aligned}$$

$$d\eta_x = \frac{\sin \phi [(P_s - P_p) dt]}{4\Delta t c/\lambda} \quad \begin{aligned} 1 \text{ WT} &\equiv 3 \text{DDA Integ} \\ (\text{XI-97}) \end{aligned}$$

$$d\eta_y = \frac{\cos \theta [(P_s - P_p) dt]}{4\Delta t c/\lambda} \quad (\text{XI-98})$$

$$d \sin \phi = \cos \phi d\phi \quad 1 \quad (\text{XI-99})$$

$$d \cos \phi = -\sin \phi d\phi \quad 1 \text{ WT} \equiv 2 \text{DDA Integ} \quad \begin{aligned} &\text{Double Precision,} \\ &\text{Mode} \end{aligned} \quad (\text{XI-100})$$

$$d \sin \theta = \cos \theta d\theta \quad 1 \quad (\text{XI-101})$$

$$d \cos \theta = -\sin \theta d\theta \quad 1 \text{ WT} \equiv 2 \text{DDA Integ} \quad \begin{aligned} &\text{Double Precision,} \\ &\text{Mode} \end{aligned} \quad (\text{XI-102})$$

* Calculation uses 9 QDPU in Single Precision ≈ 40 DDA Integrators
 2 QDPU in Double Precision ≈ 4 DDA Integrators

In terms of pure rate of routine processing possible, the result is 1 QDPU ≈ 4 DDA integrator multi-increment and double precision sinusoids raise effective speed by a supplementary factor in excess of 8 i.e., ≈ 30 times conventional DDA rate.

* The equivalents in DDA integrators is one of word times per operation ignoring QDPU superiority in algorithm multi-increment and Double precision.

CHAPTER XII
PROGRAMMABLE TRANSFER OPERATIONS OF THE QDPU
AND QDPU PROGRAMMING CODE STUDIES

12.0 INTRODUCTION - The level of versatility of the QDPU, reflected in the aerospace modal computation program studies of Chapter XI, requires a certain minimal set of programmable transfer operations. A mechanization can realize the minimal set in many equivalent ways as a result of the equivalence of members of sets of like registers and of transfer units.

The problem of developing a programming code which implies efficient mechanization involves attaining the minimal programmable set with a code of acceptable storage and decode simplicity. Mode action of the QDPU is effected by programming transfer operations and output criterions according to a program code, which is stored in the auxiliary memory associated with each QDPU. A program code will be delineated which enables programming all the transfer actions (each involving the 4 multi-transfer and 5 single transfer units). Having formed a concept of the minimal required set of programmable sets of transfer actions, the problem of determining a program code which may subsequently enable economical mechanization is investigated. The optimal code depends in considerable degree on whether core or drum memory is used. Generally, however, the factors which measure code optimality are the code length and decode complexity. The minimal code length is about 9 bits presenting, in short form, however, a more complex decode logic. On the other hand, the same program expressed in code of length 16 to 20 bits can have relatively simple decode logic but requires more flip flops. In the following analysis, the primary objective is to define the minimal set of desired programmable transfer operations of the QDPU, on which a code for minimum hardware costs may be derived.

The programmable set will be expressed in a code of intermediate length which may serve as a base for developing an equivalent code for the QDPU, efficiently adapted to the mechanization type. Each multi-transfer unit, MT, and single transfer unit, ST, must be assigned in each programmed QDPU mode action to have:

- A. a register, the contents of which are transferred by the unit (a δ or y register).
- B. a register, the contents of which determine degree and kind of transfer, i.e. (independent variable) δ register.
- C. a register or registers to which the transfer unit results are added (if added), i.e. R register (or for internal transfer) if it were mechanized a y register.

The statement of the programmable set is facilitated by the register labeling indicated in Figure 12-1. The identical function of the same register and transfer types, that is δ registers, y registers, R registers, multi-transfer units or single transfer units and assumption of complete programmability of inputs implies that the same computation can in principle be effected in a large number of equivalent programs of transfer actions. This indicates the probable existence of a simplified program code for a subset of transfer actions which call for all desired modes of computation of a perfectly general code.

Before developing a simplified QDPU program code, consider the implications of complete programmability of transfer operations. Each of 4 multi-transfer units would have independent variable selections corresponding to any one of 5 δ -register contents or full rate or zero rate, and transfer variable selections for 3 y registers. The results can go to R_1 or R_2 , or R_1 and R_2 , which implies 252 alternatives. Each of 5 single transfers would have independent variable selections corresponding to any

of 5 δ - registers or full rate or zero rate. The results can go to any one of 3 y-registers or to R_1 or R_2 , which implies 175 alternatives.

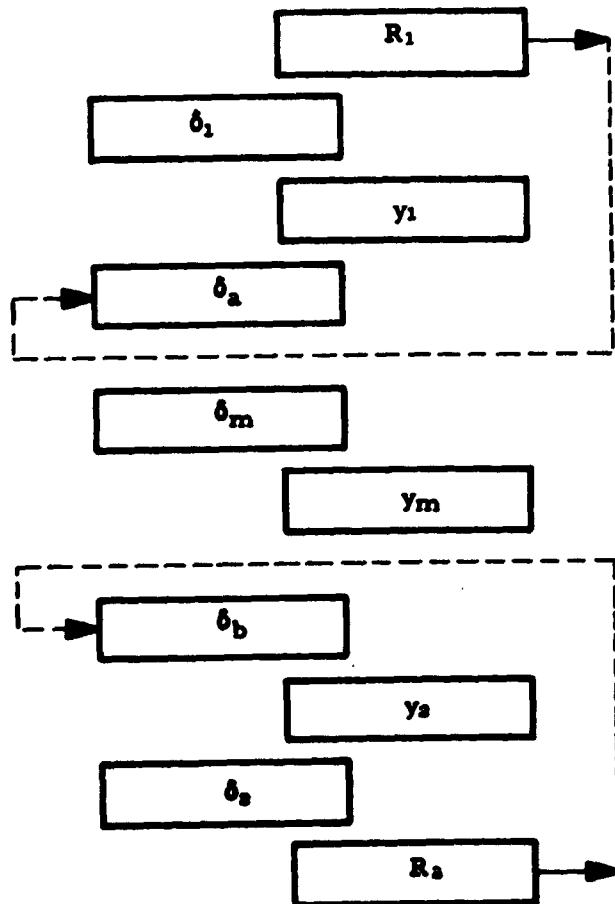


Figure 12-1. QDPU Register Labeling.

12.1 MULTI-TRANSFER AND QUOTIENT ALGORITHM MODE CODE -
 The four multi-transfer units are assigned y values which need not be programmable. The outputs of the multi-transfer units go either to R_1 or R_2 registers. No loss of computation versatility results by requiring transfers of y_1 and y_m to R_1 without programmability. The remaining transfers of y_2 and y_m should go to R_2 or R_1 programmably. The case of y_m is of use only in double precision mode; hence, it is called for in that mode only.

Thus, the multi-transfer operations T_n update R registers R_{n-1} according to the equations,

$$R_1^* = MT_1(y_1, \Delta x_1, R_{n-1}) \quad (\text{XII-1})$$

$$R_m^* = MT_2(y_m, \Delta x_2, R_{n-1}^*) \quad (\text{XII-2})$$

$$(R_1^* \text{ or } R_2^*) = MT_3(y_2, \Delta x_3, (R_1^* \text{ or } R_{n-1}^*)) \text{ according as } M = 1, 0 \quad (\text{XII-3})$$

$$(R_1^* \text{ or } R_2^*) = MT_4(y_m, \Delta x_4, (R_1^* \text{ or } R_{n-1}^*)) \text{ according as } M = 1, 0 \quad (\text{XII-4})$$

where m, M are multi-transfer unit programming bits, and Δx_n are programmed independent variables which for a Δx_n may be $\delta_1, \delta_m, \delta_2$ or x where

$$x = \begin{cases} 0 & \text{for } \Delta x_1 \text{ and } \Delta x_2 \\ \delta_a & \text{for } \Delta x_3 \text{ (if selected induces quotient algorithm)} \\ \delta_f & \text{for } \Delta x_4 \text{ (if selected induces quotient algorithm)} \end{cases}$$

All Δx selections are defined by 6 programming bits

$$\delta_1 d_1 \delta_2 d_2 \delta_3 d_3 \delta_4 d_4$$

Complete versatility of multi-transfer modes is obtained by the set of programming bits

$$mM \delta_1 d_1 \delta_2 d_2 \delta_3 d_3 \delta_4 d_4$$

These bits also determine certain actions not indicated by the R update equations:

- A. When $m = 1, M = 1$ double precision mode in which all transfers to R_1 occur.

- B. When $m = 1$, $M = 1$, y_1 instead of y_2 is used in T_3 ; thereby, obtaining double precision of R_1 transfers while not using the y_2 register for it.
- C. When $m = 1$, $M = 1$ the y_2 register is used for decision command output of R_2 channel according to sign of y_2 .
- D. When $m = 0$, $M = 1$ double precision with MT_1 , MT_3 , to R_1 and MT_3 , MT_4 to R_2 is called for (there being no decision command mode in this case).
- E. When $\Delta x_2 = \delta_a$, quotient algorithm in R_1 and when $\Delta x_4 = \delta_b$ quotient algorithm in R_2 .

12.2 SINGLE TRANSFER, DECISION RESPONSE, AND INTEGRATION ALGORITHM MODE CODE - The five single transfer units are used to update y registers and R registers programmably. The update of y registers depends on whether constants are desired in certain y registers or decision mode action is sought.

The update of R registers with single transfer is automatic for δ_a , δ_b to R_1 , R_2 respectively if non-quotient algorithm ($\Delta x_2 \neq \delta_a$, $\Delta x_4 = \delta_b$) and if $\Delta x_2 = \delta_a$ is automatic for δ_a to R_1 .

The following equations indicate single transfer unit action according to response and single transfer programming bits D , I_1 , I_2 , I_4 (it being understood that the operations indicated other than single transfer and logical multiplication with decision command variable D_m serve merely the purpose of defining applicable equations by substitutions of 1 or 0 for program bits):

$$\bar{I}_1 \Delta R_1 + I_1 \Delta y_1 = ST_1 (S_1) \quad (\text{XII-5})$$

$$\Delta y_m = D_m \odot ST_2 (I_2 \delta_1 + \delta_m \bar{I}_4) \quad (\text{XII-6})$$

$$D\delta y_1 + D\delta y_3 = I_0 \cdot ST_3 (D\delta_m + \bar{D}\delta_s) \quad (\text{XII-7})$$

$$\Delta R_1 = ST_4 (\delta_a) \text{ if } \Delta x_3 = \delta_a \quad (\text{XII-8})$$

$$\Delta R_2 = ST_5 (\delta_b) \text{ if } \Delta x_4 = \delta_b \quad (\text{XII-9})$$

Integration algorithm for y_1 differs from y_2 , y_m if an algorithm bit A is 1. The first half of the series of QDPU has lagged algorithm unless A = 1 in which case y_1 has unlagged algorithm. The second half the reverse interpretation holds. A summary of single transfer programming bits is

D I₁ I₂ I₃ A

not including $\Delta_3 d_3 = 00$, $\Delta_4 d_4 = 00$ for ST_4 and ST_5 .

12.3 SUMMARY OF QDPU PROGRAM MODE CODE - The total code is

m M $\Delta_1 d_1$ $\Delta_2 d_2$ $\Delta_3 d_3$ $\Delta_4 d_4$ D I₁ I₂ I₃ A

totaling 15 bits.

CHAPTER XIII

LOGICAL DESIGN INVESTIGATIONS OF SECOND DIFFERENCE INCREMENTAL COMPUTERS WITH CONVENTIONAL AND GENERAL (QUOTIENT) ALGORITHM

13.0 INTRODUCTION - The new concepts of multi-increment computer design with second difference computation, communication, and δ registers, developed in Chapter VII and verified in simulations described in Chapter IX, clearly present a wealth of new factors in logical design. The relatively high resolution of first differences, implied by computation of second difference single increments for the bulk of variables typically involved in internal computation rather than direct input processing, provides a basis for attaining new levels of precision where communication structure may have the same simplicity as the conventional DDA. The basic digital processing unit (generalized DDA integrator or DPU) is capable of multi-increment computation with general (quotient) algorithm for the first time and with remarkable digital processing simplicity. The question arose as to what further digital processings, based on second difference computation, appeared natural in logical design structure of the basic digital processing unit granting some license initially in consistency with system function. Results could then be adapted as basic design techniques for portions of a full scale incremental computer system, with potentially significant rewards in overall mechanization simplicity.

The first logical design effort was concentrated on the development of a multi-transfer unit especially adapted to second difference inputs. If the second difference is single increment, a derived multi-transfer unit which may be called the D^2 multiplier, offers a mechanization simplicity. This compares to the simplicity of a conventional 3 bit multiplier, capable of an M bit multi-transfer, where only the scaling of the second difference input for single increment limits the value of M . Such a unit in the DPU is capable, for example, of serial computation of a sinusoid with time as the independent variable with 20 bit resolution where steps are 10 bit increment. Conventional design methods would require a multiplier of several times the mechanization

complexity. In a simulation computer, in generation of analytic functions, the D^2 multiplier provides remarkable economy. In aerospace applications the D^2 multiplier (evaluated in Chapter VII, paragraph 7.3) presents limitations in handling external inputs. Hybridization with the conventional multiplier as appropriate restriction of application, should be a significant advance in multi-increment computer design technique. One generalization of the D^2 multiplier, which may be called the PD^2 multiplier, is utilized in the PDD³A computer (derived in paragraph 13.2). With only slight increase in complexity over the D^2 multiplier, the PD^2 multiplier incrementation of a product usually requiring two distinct multi-transfers, is possible under the same conditions on computation variables as in the case of the D^2 multiplier. A conventional two transfer mechanization for product with the same speed as the PD^2 multiplier has flip-flop requirements a factor $\frac{2M}{4} = \frac{M}{2}$ times greater complexity. A DPU with two D^2 multipliers, or two conventional multipliers, is capable of executing the general (quotient) algorithm. The D^2 multiplier offers here the first discussed mechanization saving, provided scaling of single increment second difference inputs is acceptable. The second kind of basic logical design development in second difference computation is that of natural second difference overflow for non-division algorithm, in which one register is deleted from the originally proposed mechanization. The mechanization for natural overflow offers a saving relative to the mechanization for general (quotient) algorithm in non-quotient operation. Only the general (quotient) algorithm has been simulated and verified in simulation, specifically for quotient operation, an operation which implies, because of generality, a consistent operation in the less demanding non-quotient operation (where divisor is unity or a whole word constant). The new overflow mechanization should be simulation evaluated to determine the level of roundoff error implied by an apparently reduced residue retention.

The complete logical design of the DD²A incorporating the concepts of second difference computation, communication and 6 registers for input

accumulation together with the developments of the D^2 multiplier and natural second difference overflow is presented to demonstrate the concrete mechanization of these concepts. Finally, the configuration of a QDD² A capable of full aerospace mission computations, including required input processing and internal computations at new levels of accuracy, is presented. Estimates of the flip-flop and diode requirements of the computers are presented in Table 13-6.

13.1 THE DD² A

A. Structure of DD² A integrator - The structure of the DD² A integrator is suggested by the form of the second difference of the numerical approximation $\int y dx$. Assuming trapezoidal integration, $Z_{n+1} = Z(X_{n+1}) = \sum (y_i + \frac{\Delta y_i}{2}) \Delta x_i$, where y_i represents the value of y at the beginning of the i^{th} step (so that y_i is the initial value of y). At the n^{th} step, the second difference of Z_n is given in equation (13-1).

$$\Delta^2 Z_n = y_{n+1} \Delta^2 x_n + \Delta x_n \Delta y_n + \frac{1}{2} (\Delta x_n \Delta^2 y_n + \Delta^2 x_n \Delta y_n + \Delta^2 x_n \Delta^2 y_n). \quad (13-1)$$

The term within parentheses is the first difference of $\Delta x_n \Delta y_n$, so that (13-1) can be written as (13-2).

$$\Delta^2 Z_n = y_{n+1} \Delta^2 x_n + \Delta x_n \Delta y_n + \frac{1}{2} \Delta(\Delta x_n \Delta y_n). \quad (13-2)$$

These equations, and the assumption of two-or three-valued second differences, permit the arithmetic part of the DD² A integrator to be mechanized as shown in Figure 13-1.

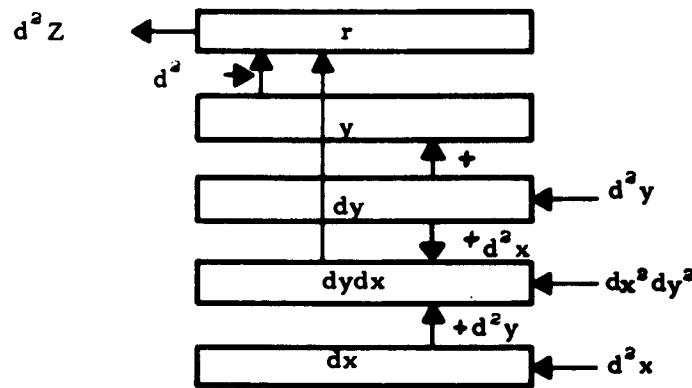


Figure 13-1. Arithmetic Mechanization of the DD²A Integrator

For rectangular integration the term $\frac{1}{2}d(dx dy)$ is not added to r . Each indicated transfer is controlled by a second differential (generated by some integrator), or is always additive, so that each transfer is simple, second differentials being only +1, -1, or 0. The over-flow mechanism, producing d^2Z , is the same as that used in the DDA to produce a three-valued dZ .

B. **DD²A Scaling Equations.** - The structure of the DD²A integrator is further clarified by scaling considerations. The existence of an X register is a convenient assumption. This is not a part of the real integrator use of the X register but assists in the proper scaling of X.

Let n_1 be the number of places in y , n_2 the number of places in dy , n_3 the number of places in dx and n_4 the number of places in X (in each case, excluding sign). Assume $n_1 + n_2 = n_3 + n_4$. This assumption is natural, and in product formation in the DDA is necessary. Then the integrator, with the x register added, has the form shown in Figure 13-2.

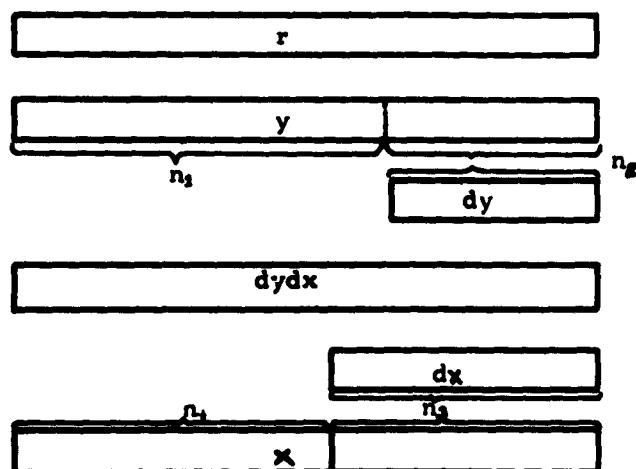


Figure 13-2. Integrator with the X Register Added

Let S_u , for any y -number u , be the scale factor of u , so that the real value of the variable u is 2^{S_u} times the machine value of u (this being bounded by 1 in absolute value). Let $w(du)$, for any differential du , be the numerical value of $(du)_{\text{max}}$ in real u -units, and let S_{du} be such that $2^{S_{du}} w(du) = 1$ u-unit; let $w(d^2u)$ and Sd^2u be analogously defined. Equations (13-3) follow at once.

$$n_1 = Sy + Sdy$$

$$n_2 = Sd^2y - Sdy \quad (\text{XIII-3})$$

$$n_3 = Sd^2x - Sdx$$

$$n_4 = Sx + Sdx$$

The proper alignment of the integrator registers requires knowledge of the weights of $(yd^2x)_{\text{max}}$, $(dy d^2x)_{\text{max}}$ and $(dx d^2y)_{\text{max}}$, as these functions enter into the arithmetic operations. These weights are given in equations (13-4).

$$(yd^2x)_{\text{max}} = 2^{Sy} \cdot 2^{Sx} \cdot 2^{-(n_3 + n_4)}$$

$$(dy d^2x)_{\text{max}} = 2^{Sy} \cdot 2^{-n_1} \cdot 2^{Sx} \cdot 2^{-(n_3 + n_4)} \quad (\text{XIII-4})$$

$$(dy d^2y)_{\text{max}} = 2^{Sx} \cdot 2^{-n_4} \cdot 2^{Sy} \cdot 2^{-(n_3 + n_4)}$$

$(v d^2x)_{\text{max}}$ is the numerical value of a full r register, which gives the scale of $d^2 Z$ as in (13-5).

$$w(d^2 Z) = 2^{Sd^2 Z} = 2^{Sy + Sx - (n_1 + n_4)}, \text{ or}$$

$$Sd^2 Z = n_3 + n_4 - (Sy + Sx). \quad (\text{XIII-5})$$

$(dy d^2 X) \text{ max} = 2^{-n_1}$. $(y d^2 X) \text{ max}$, which assures that the positioning of dy to permit its addition to y , as required by its scale, is consistent with its positioning with respect to $dx dy$, to permit the addition of $dy d^2 X$ to the latter quantity. That is, dy adds to y , n_1 places from the most significant end of y , and $dy d^2 X$ adds to $dx dy$, n_1 places from the most significant end of r . Similar relations hold for dx . This shows that the alignment in Figure 13-2 gives a proper structure for the arithmetic part of a DD² A integrator. This leads directly to a mechanization in which each register is held in a channel, on a drum, or in cores, the registers being processed serially.

Finally, from $n_1 + n_2 = n_3 + n_4$, $(yd^2 X) \text{ max} = (xd^2 y) \text{ max}$.

Although the X register is not a part of the simple DD² A integrator, the presence of the register is assumed in scaling x , so that the use of n_4 is meaningful.

- C. Input Scaling. - The integrator must have a decoding segment in addition to its integrating unit. The decoding segment performs the selection of those outputs which are its own inputs, and their fusion into three-valued second differentials, $d^2 y$ and $d^2 x$. Second differentials will now be treated as three-valued, rather than two-valued, to afford greater accuracy. Two-valued second differentials, however, permit a very significant simplification in machine design, and should be used if consistent with accuracy requirements. The fact that each integrator output is three-valued does not assure that a $d^2 y$ is also of this form, since there may be several inputs to $d^2 y$. To enable the input second differentials to be represented in this simple way, the assumption is now made that inputs to a $d^2 y$ (or $d^2 x$) are scaled as follows:

The first two appearing in the serial processing of the integrator have the same scale. Those which appear thereafter are scaled upward by powers of 2 (the third input would have twice the weight of each of the first two; the fourth, four times this weight; the fifth, eight times, and so on. The inputs to d^2y (or d^2x) now resemble a binary number, differing in that the two least significant digits have equal weight, and in that each digit may be -1 as well as +1 or 0. The decoding process for d^2y consists of identifying the inputs to d^2y , treating them collectively as a number, and then adding this number into an r register; the final three-valued carry resulting from this additive process is d^2y .

Inputs are selected for d^2y from the two d^2Z memory channels Z_1 and Z_2 , by code marks held in an extension of the y register.

The small register holding the residue of the additive process is an extension of the dy register. Similar registers are present for the decoding of d^2x . The integrator word structure is shown in Figure 13-3, along with the d^2Z channels, which are shared by all integrators.

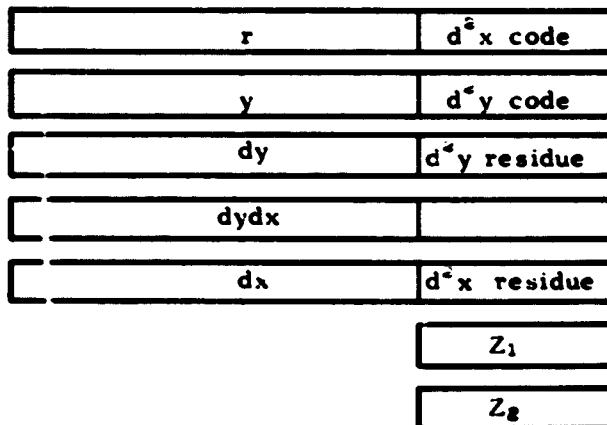


Figure 13-3. Integrator Word Structure

D. Output Modes. - The integrator is completed by adding a set of code marks in the last bit position to permit sign reversal of the output, and to make use of a decision process. A mark at this point in the dy register results in the output being set to 1 or 0 depending on whether or not y is positive. A mark in the y register results in the normal addition of the dy register to y or its replacement by 0 depending on whether the Z channels hold 1 or 0 at this point. In this way the sign of y numbers may be used to effectively replace the y number of the controlled integrator by a constant, in response to a cut-off signal. The sign reversal mark is placed in the dx register. An origin mark is also placed in the dxdy register to distinguish one integrator, as the first.

The final form of the integrator, including an associated marking channel, is given in Figure 13-4.

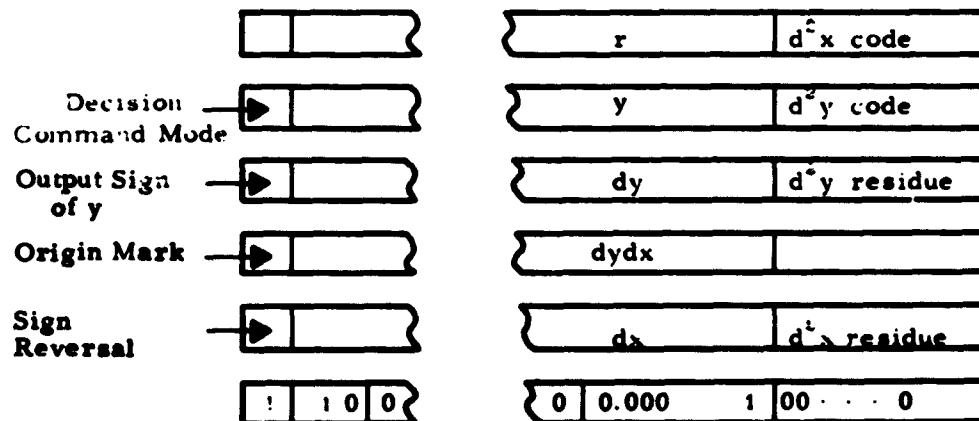


Figure 13-4. Final Form of the Integrator Including an Associated Marking Channel

E. Programming For DDA and DD² A. - The interconnection of integrators required for a given computation in the DD² A is identical to that required in DDA. The scaling equations which must be satisfied are equations (13-3) and (13-5) of 13-1B.

F. Logical Equations for DD² A.

1. Memory Structure. - The logical equations will be derived in terms of the word structure given in Figure 13-4, and the memory structure shown in Figure 13-5. The read flip-flops in the recirculating channels have the subscript "1" and the write flip-flops the subscript "2"; the r channel is one bit shorter than the other channels, this bit being added by passage through A_3 ; the channels in the mechanization to be given do not recirculate.

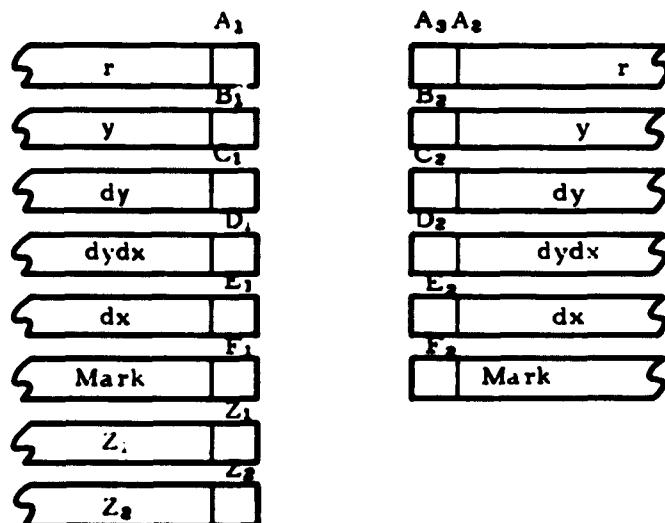


Figure 13-5. DD² A Memory Structure

2. Phases. - P_1 and P_2 are used to give the Decode and Integrate phases of the integrator, as well as the Idle and Compute controls, shown in Table 13-1. Decode passes to Integrate

TABLE 13-1

P_1	P_2	
0	0	Idle
0	1	Decode
1	1	Integrate

at $\bar{P}_1 P_2 F_1$; Integrate passes to Decode at $\bar{P}_1 F_1 F_2$. Idle is entered from integrate at the origin mark if the Stop-Go button is down, and idle passes to Decode at the origin mark if the Stop-Go button is up. The equations for P_1 and P_2 are:

$$SP_1 = \bar{P}_1 P_2 F_1$$

$$ZP_1 = F_1 F_2$$

(XIII-6)

$$SP_2 = \bar{P}_2 F_1 F_2 D_1 Go$$

$$ZP_2 = P_2 F_1 F_2 D_1 Stop$$

- G. Decode - The decoding operation for d^2y will be treated, that for d^2x being analogous. Code marks appear in B_1 , and the inputs in Z_1 and Z_2 . K_1 is used to distinguish the first input from those that follow, and Y_1 and Y_2 are the d^2y flip-flops. The values corresponding to the flip-flops states are as listed in Table 13-2.

TABLE 13-2

Y_1	Y_2	Z_1	Z_2
1	1	1	1
0		0	0
1	0	-1	1

The first input is transferred to the d^2y flip-flops, and thereafter by the scaling assumption given in 13-1C. The value held in these flip-flops is equal in weight to that of the next d^2Z decoded. When a code mark appears in B_1 the partially formed d^2y in Y_1 and Y_2 is added to the d^2Z in Z_1 and Z_2 . If the sum is +2, 0 or -2 it is passed on to the next code position, where the d^2Z has double the weight of the last. If the sum is 1 or -1, the residue bit in C_1 is used. The appearance of a 1 in C_1 records a deficiency of 1 in what was transmitted in Y_1 and Y_2 the last time a sum of 1 or -1 occurred. This deficiency is removed by sending 2 or 0, respectively; also, a 0 is written in the residue position. If a sum of 1 or -1 arises and the residue bit is a zero, 0 or -2, respectively, are transmitted and a 1 written in the residue position. The last value held in Y_1 and Y_2 is d^2y .

The terms of the logical equation related to Decode are given for d^2y and d^2x ; X_1 and X_2 are set to 1 at P_1 , F_1 , F_2 so that no d^2x code mark will result in a d^2x of 1. The term "A-B" is defined by $A-B = AB + \bar{A}B$. Also, T_5 and T_6 are used where these are given by:

$$\bar{T}_5 = (Z_1 \bar{Y}_1 (C_1 - Z_2) + \bar{Z}_1 Y_1 (C_1 - \bar{Y}_2)) + \bar{B}_1$$

$$\bar{T}_6 = (Z_1 \bar{X}_1 (E_1 - Z_2) + \bar{Z}_1 X_1 (E_1 - \bar{X}_2)) + \bar{A}_1$$

$$SK_1 = \bar{P}_1 B_1 \bar{F}_1$$

$$ZK_1 = P_1 F_1 F_2$$

$$SK_2 = \bar{P}_1 A_1 \bar{F}_2$$

$$ZK_2 = P_1 F_1 F_2$$

(XIII-7)

$$SY_1 = \bar{P}_1 (\bar{K}_1 B_1 Z_1 + K_1 \bar{Y}_1 T_5)$$

$$ZY_1 = P_1 F_1 F_2 + \bar{P}_1 (\bar{K}_1 B_1 \bar{Z}_1 + K_1 Y_1 T_5)$$

$$SY_2 = \bar{P}_1 (\bar{K}_2 B_1 Z_2 + K_2 T_5 C_1)$$

$$ZY_2 = \bar{P}_1 (\bar{K}_2 B_1 \bar{Z}_2 + K_2 T_5 \bar{C}_1)$$

$$SC_2 = \bar{P}_1 P_2 (C_1 - Z_1 - Y_1)$$

$$ZC_2 = \bar{P}_1 P_2 (\bar{C}_1 - Z_1 - Y_1)$$

$$SX_1 = P_1 F_1 F_2 + \bar{P}_1 (\bar{K}_2 A_1 Z_1 + K_2 \bar{X}_1 T_6)$$

$$ZX_1 = \bar{P}_1 (\bar{K}_2 A_1 \bar{Z}_1 + K_2 X_1 T_6)$$

$$SX_2 = P_1 F_1 F_2 + \bar{P}_1 (\bar{K}_2 A_1 Z_2 + K_2 T_6 E_1)$$

$$ZX_2 = \bar{P}_1 (\bar{K}_2 A_1 \bar{Z}_2 + K_2 T_6 \bar{E}_1)$$

$$SE_2 = \bar{P}_1 P_2 (E_1 - Z_1 - X_1)$$

$$ZE_2 = \bar{P}_1 P_2 (\bar{E}_1 - Z_1 - X_1)$$

H. Integrate - The equations defining Integrate are given in this section, and should be considered operation by operation. The sign convention is to give non-negative numbers a sign digit of zero. Q is the state $P_1 (\bar{F}_1 + \bar{F}_2)$.

1. $dy + d^2y$

K_1 is used as the carry, and is set to 1 at $P_1 F_1$ if $Y_1 = 1$.

The equations follow.

$$SK_1 = \bar{P}_1 F_1 Y_1$$

(XIII-8)

$$ZK_1 = \bar{P}_1 F_1 \bar{Y}_1 + P_1 (Y_2 - C_1)$$

$$SC_2 = Q(C_1 - K_1) + P_1 F_1 F_2 C_1$$

$$ZC_2 = Q(\bar{C}_1 - K_1) + P_1 F_1 F_2 \bar{C}_1$$

2. $dx + d^2x$ - The equations are similar to those of (1).

$$SK_2 = \bar{P}_1 F_1 X_1$$

$$ZK_2 = \bar{P}_1 F_1 \bar{X}_1 + P_1 (X_2 - E_1)$$

(XIII-9)

$$SE_2 = Q_1 (E_1 - K_2) + P_1 F_1 F_2 E_1$$

$$ZE_2 = Q_1 (\bar{E}_1 - K_2) + P_1 F_1 F_2 \bar{E}_1$$

3. $y + dy$ - K_3 is used as the carry

$$SK_3 = P_1 \bar{K}_3 B_1 C_1$$

$$ZK_3 = \bar{P}_1 F_1 + P_1 K_3 \bar{B}_1 \bar{C}_1$$

(XIII-10)

$$SB_2 = Q(K_3 - B_1 - C_1) + P_1 F_1 F_2 B_1$$

$$ZB_2 = Q(K_3 - \bar{B}_1 - C_1) + P_1 F_1 F_2 \bar{B}_1$$

4. $d^2y dx + d^2xdy + d^2y d^2x$. - Subtraction is to be accomplished by complementation and addition. This means that there must be an initial carry for each such complementation, that must absorb d^2xd^2y . If both d^2x and d^2y are -1, there will be an initial carry of 3, so that a double carry is needed. These two flip-flops are K_4 and K_5 , and their states are defined in Table 13-3. They are set initially at $\bar{P}_1 F_1$.

TABLE 13-3

K_5	K_4
0	0
0	1
1	0
1	1
	3

In the equations which follow, T_1 , T_2 and S_1 are given by:

$$T_1 = X_1 (C_1 - \bar{X}_2), T_2 = Y_1 (E_1 - \bar{Y}_2) \text{ and } S_1 = T_1 - T_2 - K_5.$$

$$SK_4 = \bar{P}_1 F_1 (X_1 Y_1 (X_2 - \bar{Y}_2) + \bar{Y}_1 X_1 \bar{X}_2 + Y_1 \bar{Y}_2 \bar{X}_1) + P_1 \bar{K}_4 (K_5 - T_1 T_2)$$

$$ZK_4 = P_1 F_1 F_2 + P_1 K_4 (K_5 - \bar{T}_1 \bar{T}_2) \quad (\text{XIII-11})$$

$$SK_5 = \bar{P}_1 F_1 X_1 \bar{X}_2 Y_1 \bar{Y}_2$$

$$ZK_5 = P_1 F_1 F_2 + P_1 K_5 (\bar{K}_4 (\bar{T}_1 + \bar{T}_2) + \bar{T}_1 \bar{T}_2)$$

S_1 is the sum $dyd^2x + dxd^2y + d^2y d^2x$, and is added to $dx dy$.

while $\frac{1}{2}S$ is added to r .

5. $dx dy + S_1$.

K_6 is the carry

$$SK_6 = P_1 \bar{K}_6 S_1 D_1 \quad SD_2 = Q (S_1 - D_1 - K_6) + P_1 F_1 F_2 D_1$$

$$ZK_6 = P_1 F_1 + P_1 X_1 \bar{S}_1 \bar{D}_1 \quad ZD_2 = Q (\bar{S}_1 - D_1 - K_6) + P_1 F_1 F_2 \bar{D}_1 \quad (\text{XIII-12})$$

6. $r + dy dx + \text{sign of } S_1$. - As r passes from A_2 to A_3 $dxdy$ is added to r . In the addition of $1/2 S_1$ to r , the sign digit of S_1 is added at $P_1 F_1 \bar{F}_2$ when the sign of r is in A_1 . At the same time, the sign digit is added to the second-last r digit while passing from A_2 to A_3 . By virtue of the sign convention used, this procedure effects the addition of $1/2 S_1$ to r . K_7 is the carry used.

$$SK_7 = P_1 \bar{K}_7 A_1 D_1$$

$$ZK_7 = \bar{P}_1 F_1 + P_1 K_7 \bar{A}_1 \bar{D}_1.$$

$$SA_2 = P_1 (A_1 - D_1 - K_7 - F_1 \bar{F}_2 S_1)$$

$$ZA_2 = P_1 (\bar{A}_1 - D_1 - K_7 - F_1 \bar{F}_2 S_1). \quad (XIII-13)$$

7. $r + y d^2 x + 1/2 S_1$. - As r passes from A_2 to A_3 , it adds to $y d^2 x$ and $1/2 S_1$. The term $T_3 = S_1 (B_1 - \bar{X}_2) (F_1 + \bar{F}_2)$ is used for $y d^2 x$, as the latter does not add to r at $P_1 \bar{F}_1 F_2$. Also, $T_4 = S_1 (\bar{F}_1 + \bar{F}_2)$ is used for $1/2 S_1$, as this does not add to r at $P_1 F_1 F_2$.

T_3 and T_4 are first added using K_8 as the carry; S_2 is the sum in this addition. S_2 and r are then added using K_8 as the carry, with S_3 as the sum. S_3 is the new r digit except at the sign position where S_3 must be completed if overflow (of either sign) occurs. The equations follow.

$$S_c = T_3 - T_4 - K_8$$

$$SK_8 = P_1 \bar{F}_1 F_2 X_1 \bar{X}_2 + P_1 \bar{K}_8 T_3 T_4$$

$$ZK_8 = \bar{P}_1 + P_1 K_8 \bar{T}_3 \bar{T}_4$$

$$SK_8 = P_1 \bar{K}_8 A_2 S_2$$

(XIII-14)

$$ZK_8 = \bar{P}_1 F_1 + P_1 K_8 \bar{A}_2 \bar{S}_2$$

$$S_3 = K_8 - A_2 - S_2$$

$$SA_3 = P_1 (S_3 (\bar{F}_1 + \bar{F}_2) - A_3 F_1 F_2)$$

$$ZA_3 = P_1 (\bar{S}_3 (\bar{F}_1 + \bar{F}_2) - \bar{A}_3 F_1 F_2).$$

If L_1 and L_2 are the expressions for positive and negative overflow, respectively, they are:

$$\begin{aligned} L_1 &= P_1 F_1 P_2 (\bar{A}_2 \bar{S}_2 (K_q - A_3) + (A_2 - S_2) A_3 K_q) \\ L_2 &= P_1 F_1 F_2 (A_2 S_2 (K_q - A_3) - (A_2 - S_2) \bar{A}_3 K_q). \end{aligned} \quad (\text{XIII-15})$$

Their sum, excluding impossible cases, is

$$L_1 + L_2 = K_q - A_3 - A_2 - S_2. \quad (\text{XIII-16})$$

8. Output. - Code marks appearing in B_1 , C_1 and E_1 affect the output communicated to other integrators. If there are no marks at this point the normal overflow given in the last section is used. $P_1 F_1 F_2 E_1$ indicates sign reversal; $P_1 F_1 F_2 C_1$ requires that the sign of y replace the normal overflow, $y \geq 0$ being sent as 1 and $y < 0$ as 0; $P_1 F_1 F_2 B_1 Z_1$ cause the normal overflow to be transmitted and $P_1 F_1 F_2 B_1 \bar{Z}_1$ force an output of zero.

Let O_1 and O_2 represent the output as in Table 13-4. Then

TABLE 13-4.

O_1	O_2	
1	1	1
0		0
1	0	-1

O_1 and O_2 are defined as follows.

$$\begin{aligned}
 S0_1 &= P_1 F_1 F_2 (\bar{B}_1 \bar{C}_1 (L_1 + L_2) + B_1 Z_1 (L_1 + L_2) + C_1 \bar{B}_2) \\
 Z0_1 &= P_1 F_1 F_2 (\bar{B}_1 \bar{C}_1 \bar{L}_1 \bar{L}_2 + B_1 Z_1 \bar{L}_1 \bar{L}_2 + B_1 \bar{Z}_1 + C_1 B_2) \\
 S0_2 &= P_1 F_1 F_2 (L_1 \bar{E}_1 + L_2 E_1) \\
 Z0_2 &= P_1 F_1 F_2 (L_2 E_1 + L_1 E_1)
 \end{aligned} \tag{XIII-17}$$

9. Complete Logical Equations. - The unsimplified logical equations are now listed.

$$\begin{aligned}
 T_1 &= X_1 (C_1 - \bar{X}_2) & S_1 &= T_1 - T_2 - K_1 \\
 T_2 &= Y_1 (E_1 - \bar{Y}_2) & S_2 &= T_3 - T_4 - K_2 \\
 T_3 &= X_1 (B_1 - \bar{X}_2) (F_1 + \bar{F}_2) & S_3 &= S_2 - A_2 - K_3 \\
 T_4 &= S_1 (\bar{F}_1 + \bar{F}_2) & Q &= P_1 (\bar{F}_1 + \bar{F}_2) \\
 \bar{T}_5 &= (Z_1 \bar{Y}_1 (C_1 - Z_2) + \bar{Z}_1 Y_1 (O_1 - \bar{Y}_2)) + \bar{B}_1 \\
 \bar{T}_6 &= (Z_1 \bar{X}_1 (E_1 - Z_2) + \bar{Z}_1 X_1 (E_1 - \bar{X}_2)) + \bar{A}_1 \\
 SK_1 &= \bar{P}_1 \bar{F}_1 B_1 + \bar{P}_1 F_1 Y_1 \\
 ZK_1 &= P_1 F_1 F_2 + \bar{P}_1 F_1 \bar{Y}_1 + P_1 (Y_2 - C_1) \\
 SK_2 &= \bar{P}_1 \bar{F}_1 A_1 + \bar{P}_1 F_1 X_1 \\
 ZK_2 &= P_1 F_1 F_2 + \bar{P}_1 F_1 \bar{X}_1 + P_1 (X_2 - E_1) \\
 SK_3 &= P_1 \bar{K}_3 B_1 C_1 \\
 ZK_3 &= \bar{P}_1 F_1 + P_1 K_2 \bar{B}_1 \bar{C}_1 \\
 SK_4 &= \bar{P}_1 E_1 (X_1 Y_1 (X_2 - \bar{Y}_2) + \bar{Y}_1 X_1 \bar{X}_2 + \bar{X}_1 Y_1 \bar{Y}_2) + P_1 \bar{K}_4 (K_4 - T_1 T_2)
 \end{aligned} \tag{XIII-18}$$

$$\begin{aligned}
ZK_4 &= P_1 F_1 P_2 + P_1 K_4 (K_5 - \bar{T}_1 \bar{T}_2) \\
SK_5 &= \bar{P}_1 F_1 X_1 \bar{X}_2 Y_1 \bar{Y}_2 \\
ZK_5 &= P_1 F_1 F_2 + P_1 K_5 (\bar{K}_4 (\bar{T}_1 + \bar{T}_2) + \bar{T}_1 \bar{T}_2) \\
SK_6 &= P_1 \bar{K}_6 S_1 D_1 \\
ZK_6 &= \bar{P}_1 F_1 + P_1 K_6 \bar{S}_1 \bar{D}_1 \\
SK_7 &= P_1 \bar{K}_7 A_1 D_1 \\
ZK_7 &= \bar{P}_1 F_1 + P_1 K_7 \bar{A}_1 \bar{D}_1 \\
SK_8 &= P_1 \bar{F}_1 F_2 X_1 \bar{X}_2 + P_1 \bar{K}_8 T_1 T_4 \\
ZK_8 &= \bar{P}_1 + P_1 K_8 \bar{T}_3 \bar{T}_4 \\
SK_9 &= P_1 \bar{K}_9 A_2 S_2 \\
ZK_9 &= \bar{P}_1 F_1 + P_1 K_9 \bar{A}_2 \bar{S}_2 \\
SY_1 &= \bar{P}_1 (\bar{K}_1 B_1 Z_1 + K_1 \bar{Y}_1 T_1) \\
ZY_1 &= P_1 F_1 F_2 + \bar{P}_1 (\bar{K}_1 B_1 \bar{Z}_1 + K_1 Y_1 T_1) \\
SY_2 &= \bar{P}_1 (\bar{K}_1 B_1 Z_2 + K_1 T_1 C_1) \\
ZY_2 &= \bar{P}_1 (\bar{K}_1 B_1 \bar{Z}_2 + K_1 T_1 \bar{C}_1) \\
SX_1 &= P_1 F_1 F_2 + \bar{P}_1 (\bar{K}_9 A_1 Z_1 + K_9 \bar{X}_1 T_1) \\
ZX_1 &= \bar{P}_1 (\bar{K}_9 A_1 \bar{Z}_1 + K_9 X_1 T_1) \\
SP_1 &= \bar{P}_1 P_2 P_1 \\
ZP_1 &= F_1 F_2 \\
SP_2 &= \bar{P}_2 F_1 F_2 D_1 Go \\
ZP_3 &= P_2 F_1 F_2 D_1 Stop
\end{aligned}$$

$$\begin{aligned}
L_1 &= P_1 F_1 F_2 (\bar{A}_2 \bar{S}_2 (K_0 - A_3) + (A_2 - S_2) A_3 K_0) \\
L_2 &= P_1 F_1 F_2 (A_2 S_2 (K_0 - A_3) + (A_2 - S_2) \bar{A}_3 \bar{K}_0) \\
S0_1 &= P_1 F_1 F_2 (\bar{B}_1 \bar{C}_1 (L_1 + L_2) + B_1 Z_1 (L_1 + L_2) + C_1 \bar{B}_2) \\
Z0_1 &= P_1 F_1 F_2 (\bar{B}_1 \bar{C}_1 \bar{L}_1 \bar{L}_2 + B_1 Z_1 \bar{L}_1 \bar{L}_2 + B_1 \bar{Z}_1 + C_1 B_2) \\
S0_2 &= P_1 F_1 F_2 (L_1 \bar{E}_1 + L_2 E_1) \\
Z0_2 &= P_1 F_1 F_2 (L \bar{E}_1 + L_1 E_1) \\
SA_r &= \bar{P}_r A_1 + P_r (\bar{P}_1 A_1 + P_1 (A_1 - D_1 - K_r - F_1 \bar{F}_2 S_1)) \\
ZA_r &= \bar{P}_r \bar{A}_1 + P_r (\bar{P}_1 \bar{A}_1 + P_1 (\bar{A}_1 - D_1 - K_r - \bar{F}_1 S_1)) \\
SA_c &= \bar{P}_r A_r + P_r (\bar{P}_1 A_r + P_1 (S_r (\bar{F}_1 + \bar{F}_2) - A_3 F_1 F_r)) \\
ZA_c &= \bar{P}_r \bar{A}_r + P_r (\bar{P}_1 \bar{A}_r + P_1 (\bar{S}_r (\bar{F}_1 + \bar{F}_2) - \bar{A}_1 F_1 F_r)) \\
SB_r &= \bar{P}_r B_1 + P_r (\bar{P}_1 B_1 + Q (K_0 - B_1 - C_1) + P_1 F_1 F_2 B_1) \\
ZB_r &= \bar{P}_r \bar{B}_1 + P_r (\bar{P}_1 \bar{B}_1 + Q (K_0 - \bar{B}_1 - C_1) + P_1 F_1 F_2 \bar{B}_1) \\
SC_r &= \bar{P}_r C_1 + P_r (\bar{P}_1 (\bar{C}_1 - Z_1 - Y_1) + Q (\bar{C}_1 - K_1) + P_1 F_1 F_r \bar{C}_1) \\
ZC_r &= \bar{P}_r \bar{C}_1 + P_r (\bar{P}_1 (\bar{C}_1 - Z_1 - Y_1) + Q (\bar{C}_1 - K_1) + P_1 F_1 F_2 \bar{C}_1) \\
SD_r &= \bar{P}_r D_1 + P_r (Q (S_1 - \bar{D}_1 - K_r) + P_1 F_1 F_2 \bar{D}_1) \\
ZD_r &= \bar{P}_r \bar{D}_1 + P_r (Q (S_1 - \bar{D}_1 - K_r) + P_1 F_1 F_2 \bar{D}_1) \\
SE_r &= \bar{P}_r E_1 + P_r (\bar{P}_1 (E_1 - Z_1 - Y_1) + Q (E_1 - K_2) + P_1 F_1 F_2 E_1) \\
ZE_r &= \bar{P}_r \bar{E}_1 + P_r (\bar{P}_1 (\bar{E}_1 - Z_1 - X_1) + Q (\bar{E}_1 - K_2) + P_1 F_1 F_2 \bar{E}_1) \\
SF_r &= F_1 \\
ZF_r &= F_1
\end{aligned}$$

13.2 THE PDD²A AND QDD²A MECHANIZATIONS

- A. Introduction - The DD²A described in the previous sections was based on a computational process which provided a basic improvement in rate-handling ability and accuracy with respect to that of the DDA based on second difference computation. Incremental computer designs which further exploit this basic design approach to obtain increased processing efficiency and computation precision are the PDD²A and QDD²A computers described below.
- B. PDD²A - If the computation involves a considerable number of multiplications (vector resolutions), or if many constants appear in the equations to be solved, a sixth register may be added to the DD²A integrator. This will be the x register referred to in 13-1B, and its presence will effect a reduction of the number of integrators required in the sort of computation mentioned.

First, the generation of a product, $x y$, in a DD²A is accomplished by a double summation of its second difference. This second difference, at the n^{th} step of the process, is given in equation 13-6. Except for the coefficients, the right hand side of 13-19 differs from the right hand side of

$$\Delta^2(xy)_n = y_n + \Delta^2 X_n + x_{n+1} \Delta^2 y_n + 2\Delta X_n \Delta y_n + \Delta^2 X_n \Delta y_n + \Delta^2 y_n \Delta X_n + \Delta^2 y_n \Delta^2 X_n \quad (13-19)$$

equation (13-1) of 13-1A only by the presence of $x_{n+1} \Delta^2 y_n$. With the x register available, this term may be added to r by a transfer controlled by $\Delta^2 y$. The output of the unit shown in Figure 13-6 is the second differential of the product xy , and a PDD²A is a machine consisting of such units. The decoding

procedure is the same as that in the DDA, and the scaling equations are still (13-3) and (13-5) of 13-1B.

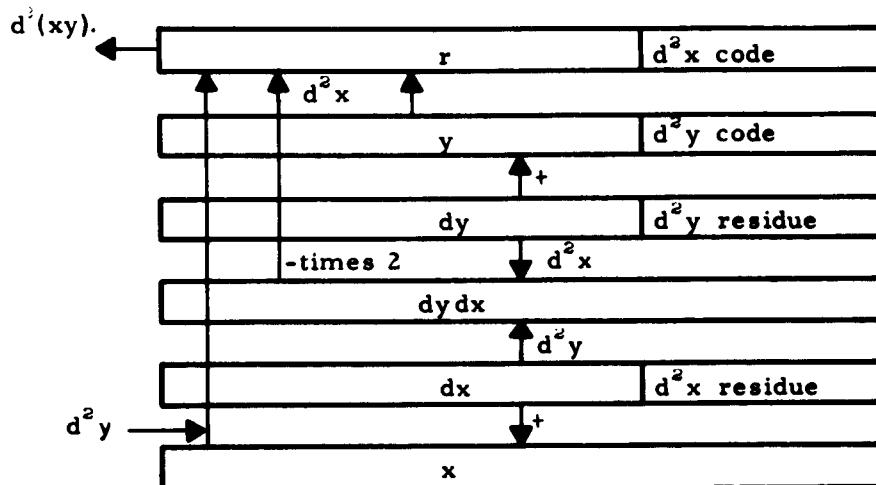


Figure 13-6. PDD²A Output of the Second Differential Product of Xy.

Only one PDD²A unit is required for product generation as opposed to two DD²A units, and the use of a single r number instead of two improves the accuracy of the computation.

The presence of the sixth register also permits the introduction of scaling factors by a modification of the overflow mechanism. Normally, the DD²A integrator output is developed by noting whether the r number, after the addition of $y d^2 x + dx dy + 1/2 d(dx dy)$, is more than 1/2 or less than -1/2. In the first case the output is 1, in the second -1, and if neither case obtains the output is 0; further, the output is subtracted from r to yield the r number for the next step. This same process may be carried out with respect to $\pm u/2$ instead of $\pm 1/2$, where u is a constant held in the x register. The output will now represent $\frac{d(ydx)}{u}$ rather than $d(ydx)$, and arbitrary scale factors may be introduced in this way. The transfer of the x

number (u) to r is now controlled by $-d^2z$, where d^2z is the integrator output on the last cycle, and dx does not add to the x register.

Thus, the sixth register results in the saving of one DD^2A integrator for each product generation, and of one DD^2A integrator for each multiplication by a constant, when not an integral power of 2.

- C. Elementary QDD^2A - The addition of a seventh register, as well as a third input; yields a unit capable of quotient generation; this unit is shown in Figure 13-7.

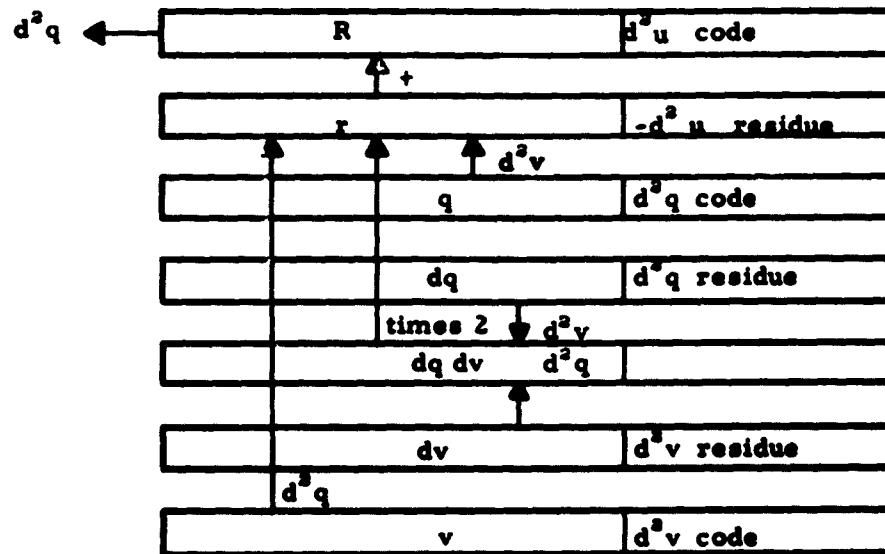


Figure 13-7. Elementary QDD^2A

Here, d^2u and d^2v are inputs, and the function of the unit is to produce the second differential of u/v . The unit forms $d(qv-u)$ in r and $(qv-u)$ in R , where q is the machine value of the quotient. The overflow, d^2q , is generated with respect to $\pm v/2$, as in the last section, although v is now a variable; in particular $d^2q = \text{sgn}R \text{ sgn}v U(2/R/-v/)$, where U is the unit step function. The output is fed back negatively to alter dq as required by $qv-u$, which reflects the error in q in representing the true quotient.

A QDD²A is a machine generalized from the elementary QDD²A unit. By coding, such a unit can be used for product generation or scaled integration, as well as quotient generation.

A hardware estimate is given in the table below.

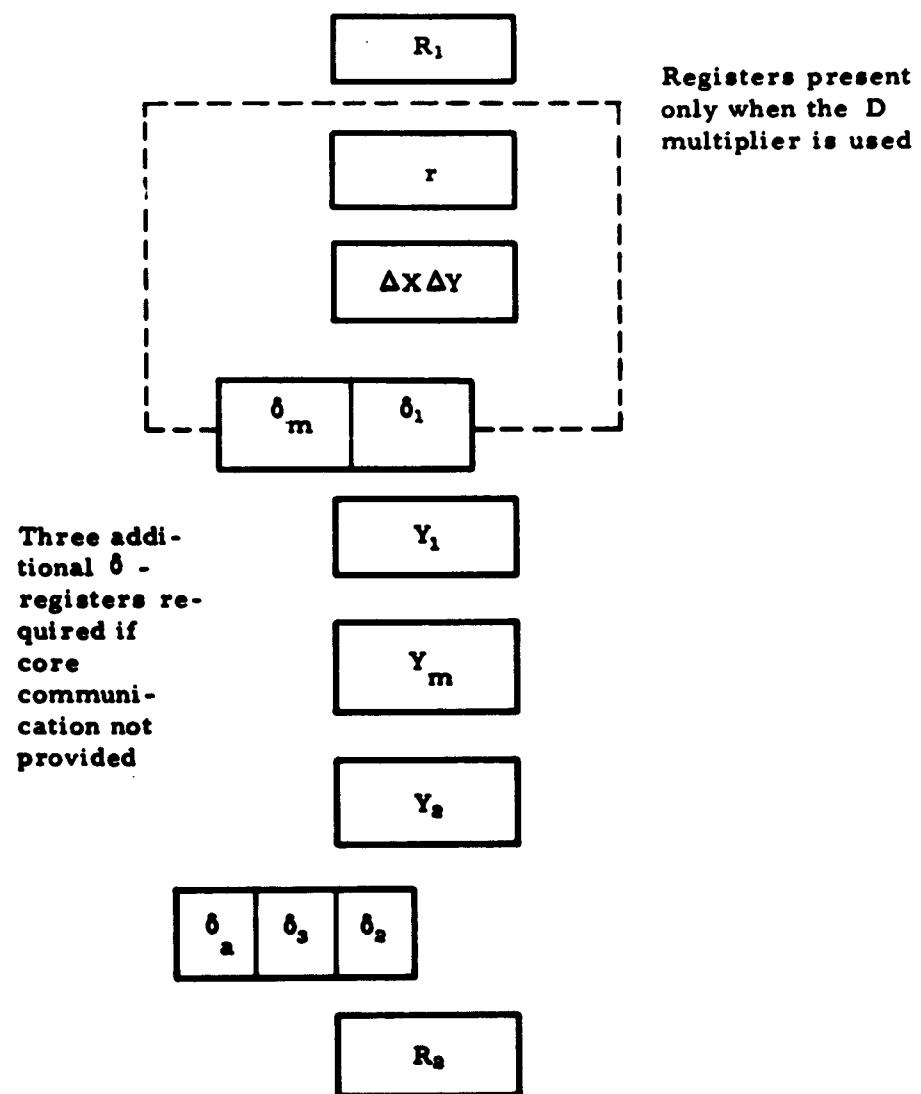
TABLE 13-5. HARDWARE ESTIMATE

<u>Unit</u>	<u>Section</u>	<u>Diodes</u>	<u>Flip/Flops</u>	<u>Channels</u>
DD ² A	2	1200	29	6
PDD ² A	3. 2	1350	34	7
QDD ² A*	3. 3	1600	40	8

*Elementary QDD²A

D. Mechanization and Functional Features of the Proposed and Alternate QDD²A Computers.

Register Configuration



Processing capability (assuming state-of-the-art 400,000 bit/sec) of Computer No. 1: Full Aerospace Mission DDA

Proposed:

1. A multi-iteration rate, multi-increment DDA for full aerospace mission which is capable of parallel computations with quotient algorithm.
2. Simultaneous thrust cutoff and strap-down computations at 1600 iter/sec in a program of 256 DDA integrators.
3. Single precision (3 bit increment), double precision (6 bit) programmable.

Computer No. 2: Aerospace DDA (Reduced Task, i.e., No High Rate Input Processing)

Proposed:

1. Where the DDA is not allocated thrust cutoff and strap-down computations an appropriate computer executes a 300 DDA integrator program at 213 iter/sec.
2. Quotient algorithm and single precision (bit increment) double precision (6 bit) programmable using D^2 multipliers and conventional multipliers.

**Flip-flop and Core Requirement Estimates (the latter optional) of
the Proposed and Alternate Incremental Computers**

<u>Sub-Units</u>	<u>Computer No. 1 (Proposed)</u>	<u>Computer No. 2</u>
12 word core storage for communication and 7 channels on drum	Conventional communication and 10 channels on drum	15 word core storage for communication and 12 channels on drum
Conventional 3 bit multi-transfers	28 F/F (4 units)	14 F/F (2 units)
D ² Multipliers	0	0
Second order integrations	4	4 (2 units)
Five Y and R channels	15 (2 iter rates)	4
Three Y registers update	3	3
8 channels and update (core storage of ΔX's of 6m, 6a, 6g in appropriate case)	10 (2 channels)	10 (1 iteration rate)
Program decode	15	15
QDPU output logic	4	4
Diode Count = 40 × Flip-Flop Count	<hr/> 79 flip/flops and 12 words cores	<hr/> 72 flip/flops and 15 words cores
		<hr/> 84 flip/flops

CHAPTER XIV

THE FUTURE ROLE OF THE INCREMENTAL COMPUTER IN FULL SCALE AEROSPACE COMPUTER SYSTEMS AND PROPOSED STUDY EFFORTS

14.0 LONG-TERM DESIGN GOALS AND REMAINING PROBLEMS IN THEIR FULL ACHIEVEMENT - This contract study (see Chapter XV) accomplishes the development of aerospace incremental computer design techniques (exemplified in the QDD²A) which enable real time computation of large programs by a computer mechanization of assigned complexity at new levels of accuracy for variables having the degree of continuity ordinarily assigned to or considered feasible for DDA type computers. The ordinary concept of a GP-DDA system with a relatively complex costly GP being required for a full aerospace mission actually resides in the fact that certain real time computations involve system variables which are only piece-wise continuous (apart from communication link data inputs which are assumed specially provided for). There had been a prevailing concept of the inability of conventional incremental computer design, and programming techniques to handle all the problematic routines involving variables with step changes and singularities implying only limited use of the DDA. In contrast, it is proposed here that the development of the appropriate incremental computer design, and programming techniques can essentially eliminate the GP, as such, in that a GP of the cost level ordinarily assumed in full mission aerospace applications is not required.

A conviction that the DDA techniques can be accomplished in an efficient mechanization implies that a significant increase in computation capability for given mechanization complexity will result in the overall computer system. An intermediate accomplishment would be the development of a GP-DDA system in which the GP is highly simplified in mechanization and with primarily low-rate supervisory capability over the DDA. The latter actually executing >95% to >99% of the computations(instead of 15% to 30%).

As shown in paragraph 11.4D, an incremental computer can have a speed advantage over a conventional GP of 6 to 1 for the same bit rates, hence, apart from ability to handle variables with discontinuities, the GP is not only basically more costly, but also, slower than the incremental computer. The degree, to which the GP can be simplified while retaining ability to handle variables with discontinuities, is therefore the prime question. That 100% handling of the piece-wise continuous variables, which make up aerospace guidance and control computations, is possible in a hybridized incremental computer is given support by the analyses of XI B 4b, C8, C6. The first two references present new techniques and approaches to technique development for handling this problem while the last two indicate basic application of conventional decision action efficiently generalized to the QDD⁸A.

The problematic computations involving isolated singularities are those where high accuracy must be maintained over long term operation. These computations typically involve singularities resulting from properties of coordinate systems. In principle these computations are resolved directly by better selection of a coordinate system. There are cases where this is not permitted in the fullest sense, for example: geographic coordinates, presenting discontinuity problems, are designated for display purposes and are necessary for gravity computation. An incremental computer must therefore be able to handle the geographic coordinate problem, when coordinates differ significantly from the well defined coordinates of singularities that required accuracy is maintained. It is believed that, provided this particular example problem can be overcome in a proposed study effort, then the solution of any other problem is relatively straightforward.

The second class of design problems proposed for further study are those of further unification by system and logical design analysis and DDA simulation of the many computation algorithm and digital processing discoveries

made during this contract effort. While quantitative computation analyses and simulations demonstrate that the QDD²A is a remarkable step in computer design, it is believed the refinement of the developed design techniques, incorporated in the QDD²A, can offer significant increases in computation capability and reduction in mechanization complexity.

14.1 BRIEF SUMMARY OF PROPOSED STUDY EFFORTS

- A. Completion of Simulation Evaluation and Analysis of All Developments of Phase II and Proposed Analytical Efforts**
 - 1. Evaluation and comparison of alternative QDD²A algorithms**
 - 2. Evaluation and comparison of alternative DD²A algorithm**
 - 3. Digital Stieltjes algorithm for near conventional single increment DDA and generalization to multi-increment computers**
 - 4. Overflow inhibitor and pulse stream transducer**
 - 5. Evaluation of singularity, discontinuity pass programs of adapted mechanizations**
- B. Logical Design Investigations**
 - 1. Optimized communication mechanization for large problem incremental computers and GP-DDA with atrophied GP**
 - 2. Extension of multi-increment arithmetic unit design studies for band limited variables**
 - 3. Programmable single, double precision modes involving the D² multiplier**
 - 4. System evaluation and optimization by execution of modal mechanization, register, and arithmetic unit costs relations analysis for minimal hardware count at assigned computation capability.**

C. Analytical Investigation of Improved Processing Complementation Structure for GP and DDA in GP-DDA Computer System with atrophied GP

- 1. Immediate goal is to make possible inherent but not fully attained computation rate superiority of DDA over GP (which is 4 or 6 to 1) in > 95 percent of aerospace programs without excessive supervision of DDA by GP**
- 2. Ultimate goal of greatly reducing or essentially eliminating the major GP hardware cost**
- 3. Refine the quantitative computation capability formulations of Phase II in light of advances in digital Stieltjes integration**

D. Further Analysis of the Pulse Stream Analog to Digital Converter and Overflow Inhibitor for Improved Computation Accuracy at Low Rate Phases of Inputs.

CHAPTER XV
BRIEF SUMMARY OF ACCOMPLISHMENTS OF THE
HSDDA STUDY EFFORT

**15.0 DEVELOPMENT OF FULL SCALE INCREMENTAL COMPUTER
SYSTEM (QDD²A) FOR FULL AEROSPACE MISSION**

- A. Total program includes: programmable input processing (for simultaneous thrust cutoff and strap-down navigation) with multi-increment accuracy (assuming modest clock rates) at 1600 iterations/sec, and internal computations at 100 iterations/sec for a 256 DDA integrator program.**
- B. Communication hardware simpler than a conventional single increment DDA of same capacity (an invention).**
- C. Computation with programmable single precision (3 bit) and double precision (6 bit) transfer action (an invention).**
- D. Multi-increment quotient algorithm (an invention) for orbital and re-entry computations.**
- E. The total program exceeds the computation capacity of four 3 bit increment DDA computers and executes 6 bit increment computation programmably for error sensitive routines.**
- F. The total program, combined with a slow multiplier general purpose computer, provides simpler system mechanization than existing aerospace computers which generally have one-half or less the computation capacity. A proposed QDD²A mechanization has a 12 word core memory for communication and input absorption and 79 flip-flops (30 percent less than the strap-down processor section constructed).**

The advantage over existing GP-DDA systems can be further increased by proposed further studies directed toward utilizing the inherent but unrealized computation superiority of DDA over GP in 90 percent of aerospace program routines.

15.1 DEVELOPMENT OF CONCEPTS FOR DESIGN OF MULTI-INCREMENT COMPUTER WITH SIMPLIFIED MECHANIZATION - An epochal breakthrough in multi-increment computer design has been made for the incremental computation of the band limited variables which characterize typical DDA computations. The DDA integrators (or generalized integrators) have outputs which represent second differentials rather than first differentials. The communication of second differentials is mechanized in the manner of first differentials in a conventional DDA. Developments during Phase II which have exploited the new concepts are:

- A. Single increment communication for a multi-increment computer attaining a new level of communication mechanization simplicity.
- B. Quotient algorithm computation with multi-increment accuracy not approximated by any previously existing DDA.
- C. Simplified arithmetic unit design for multi-increment computation.
- D. Second order integration algorithm realization in simplified mechanization.

15.2 DEVELOPMENT OF QUOTIENT ALGORITHM FOR MULTI-INCREMENT COMPUTATION - All previous quotient algorithm computers have been limited to basically single increment accuracy although this increment might have a variable scale. The technical design problems which have inhibited the development of a multi-increment algorithm have been overcome. The newly developed algorithm has mechanization cost comparable to that of variable increment algorithm which is less accurate.

15.3 DEVELOPMENT OF A NEW MULTI-TRANSFER UNIT (THE D² MULTIPLIER) WITH SIMPLIFIED MECHANIZATION (FOR COMPUTERS WITH SECOND DIFFERENCE COMMUNICATION OF BAND LIMITED VARIABLES)

- A.** A breakthrough in incremental arithmetic unit capability for given complexity has been made. Previously presumed inherent complexity levels have been lowered for the important class of band limited variables typically involved in internal computations or programmed on simulation incremental computers.
- B.** The D² multiplier unit can perform product calculation with higher accuracy than two conventional multiplier units for 3 bit transfer but the new unit costs the same as a single one. The D² multiplier unit can perform many bit increment computations depending on the scaling properties of the variables; there are example calculations in which the simple unit can exceed speed and accuracy of a high performance general purpose computer.

15.4 A BREAKTHROUGH IN ACCURACY IN CONVENTIONAL TYPE SINGLE INCREMENT DDA BY MODIFICATION TO EXECUTE A NEW DIGITAL STIELTJES INTEGRATION ALGORITHM CONTAINS:

- A.** Integration with respect to independent variables other than time (Stieltjes integration) constitutes >75% of all DDA Computations including division, reciprocal, product, input processing.
- B.** Incorrect digital Stieltjes integration algorithm has been determined the major error source in single increment DDA in these computations, including reciprocal calculation.
- C.** By modest elaboration in mechanization of the conventional DDA new accuracy levels are attainable.

D. Doppler damped inertial navigation by near conventional DDA
is for the first time attainable.

**15.5 PRELIMINARY DEVELOPMENT OF INCREMENTAL COMPUTERS
OF INTERMEDIATE COMPLEXITY FOR SPECIAL APPLICATIONS WHERE
NO INPUT PROCESSING IS REQUIRED.**

- A. Certain airborne and aerospace applications require relatively high computation capability but do not require input processing in the DDA though perhaps in the GP of a GP-DDA system. In these applications a simpler mechanization than the full scale QDD²A is feasible.
- B. The design combination of D² multipliers and quotient algorithm (the latter with programmable multi-and single-transfer) provides extraordinary computation features:
 1. Two D² multiplier units (costing the same as a 3 bit transfer unit) can in many calculation routines do the work* of six DDA computers each with 3 bit (or in certain cases more) increment accuracies.
 2. Two D² multipliers and 2 single transfer units in other prevalent computations routines can provide, in parallel computation, the work of four DDA computers.

*The quotient algorithm has utility for whole word scaling as well as division, in which case effective performances stated in (1) may be reduced to that of three DDA computers.

**15.6 PRELIMINARY THEORY OF PULSE STREAM ANALOGUE TO
DIGITAL CONVERTER ERROR STRUCTURE AND DIGITAL STIELTJES
INTEGRATION FOR HIGH RATE INPUT PROCESSING BY A SINGLE
INCREMENT DDA**

- A. Preliminary analysis and simulation results for roundoff reduction processes of overflow inhibition and digital Stieltjes integration appear applicable to the pulse stream transducer as well as single increment DDA systems.**
- B. Input processing algorithm based on these results for single increment input processings can provide improved performance with modest hardware modifications.**

		UNCLASSIFIED	
Aeronautical Systems Division, AF Avionics Laboratory, Electronic Technology Division, Wright-Patterson Air Force Base, Ohio.	Rpt Nr ASD-TDR 63-158. DIGITAL DIFFERENTIAL ANALYZER, Final Report, 401 p. incl illus., tables, 4 refs.	1. Computers 2. Multi-increment computer 3. Aerospace Mission 4. Digital Computers	1. AFSC Project 4421, Task 50920 II. Contract AF 33(616) -6939
This study deals with the detailed investigation of a general class of incremental computation techniques, together with techniques for their physical realization. Emphasis is placed on maximizing computation capability while minimizing hardware. Major developments resulting from this study, include: (1) Development of new incremental computation methods	III. Litton Systems, Inc, Woodland Hills, Calif.	1. Computers 2. Multi-increment computer 3. Aerospace Mission 4. Digital Computers	1. AFSC Project 4421, Task 50920 II. Contract AF 33(616) -6939 III. Litton Systems, Inc, Woodland Hills, Calif.
(over)	IV. H. Banbrook V. Report No. 2305-27 VI. Not avail fr OTS VII. In ASTIA collection	1. Computers 2. Multi-increment computer 3. Aerospace Mission 4. Digital Computers	1. AFSC Project 4421, Task 50920 II. Contract AF 33(616) -6939 III. Litton Systems, Inc, Woodland Hills, Calif. IV. H. Banbrook V. Report No. 2305-27 VI. Not avail fr OTS VII. In ASTIA collection
	UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED

(over)

for integration, multiplication, and division for fullscale aerospace mission computers; (2)Development of a second order DDA for use with band-limited variables; and (3)Development of a true Stieljes integration technique.

UNCLASSIFIED

UNCLASSIFIED